US005710901A

United States Patent [19]

Stodghill et al.

[11] Patent Number: 5,710,901

[45] Date of Patent: Jan. 20, 1998

[54] **METHOD AND APPARATUS FOR VALIDATING DATA ENTERED BY A USER**

[75] Inventors: **Scott A. Stodghill**, Evergreen; **Llad Mattie**, Englewood, both of Colo.

[73] Assignee: **TCI Summitrak of Texas, Inc.**, Englewood, Colo.

[21] Appl. No.: **581,778**

[22] Filed: **Dec. 29, 1995**

[51] **Int. Cl.**[6] **G06F 3/00**

[52] **U.S. Cl.** **395/339**; 395/185.1; 395/764; 371/48

[58] **Field of Search** 395/339, 185.1, 395/183.22, 326-358, 939, 764, 185.01; 371/48

[56] **References Cited**

U.S. PATENT DOCUMENTS

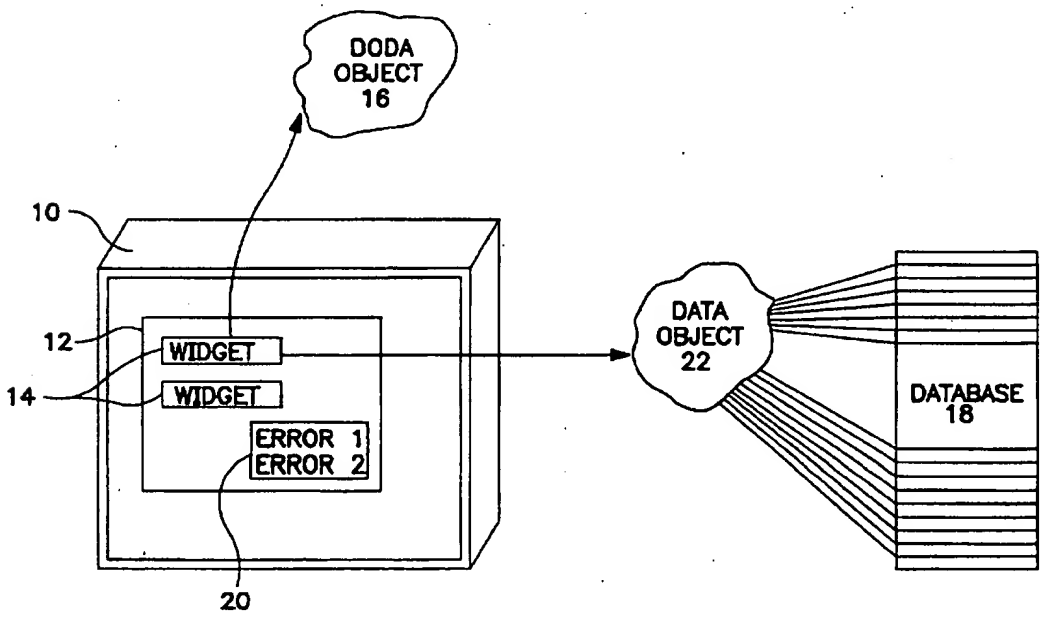
4,459,678	7/1984	McCaskill et al.	364/900
4,484,304	11/1984	Anderson et al.	364/900
5,021,973	6/1991	Hernandez et al.	395/765
5,179,657	1/1993	Dykstal et al.	395/161
5,208,907	5/1993	Shelton et al.	395/766
5,325,290	6/1994	Cauffman et al.	364/401
5,544,303	8/1996	Maroteaux et al.	395/161
5,613,065	3/1997	Ishibashi et al.	395/185.01

Primary Examiner—Raymond J. Bayerl
Assistant Examiner—A. Katbab
Attorney, Agent, or Firm—Baker & Botts, L.L.P.

[57] **ABSTRACT**

A method and apparatus for validating data entered into a data processing system. A plurality of windows is provided, each including at least one widget area into which data is entered by the user. A "doda object" is associated with each widget and contains keys identifying where the data is to be stored in a database, as well as member functions which operate to format the entered data for storage and format information retrieved from the database for display in the widget. During the validation procedure, all of the data associated with each of the widgets in a window are first examined for errors in conjunction with its associated doda object. In addition, a second validation step is performed in which the data associated with certain widgets are compared to those of other widgets in the window, in accordance with an algorithm selected by the user, to detect the presence of errors resulting from data entered inconsistently between widgets. If any errors are detected, an error message is displayed within a validation window. The widget in which the error was detected may then be selected by the user by double-clicking on the error message.

6 Claims, 3 Drawing Sheets



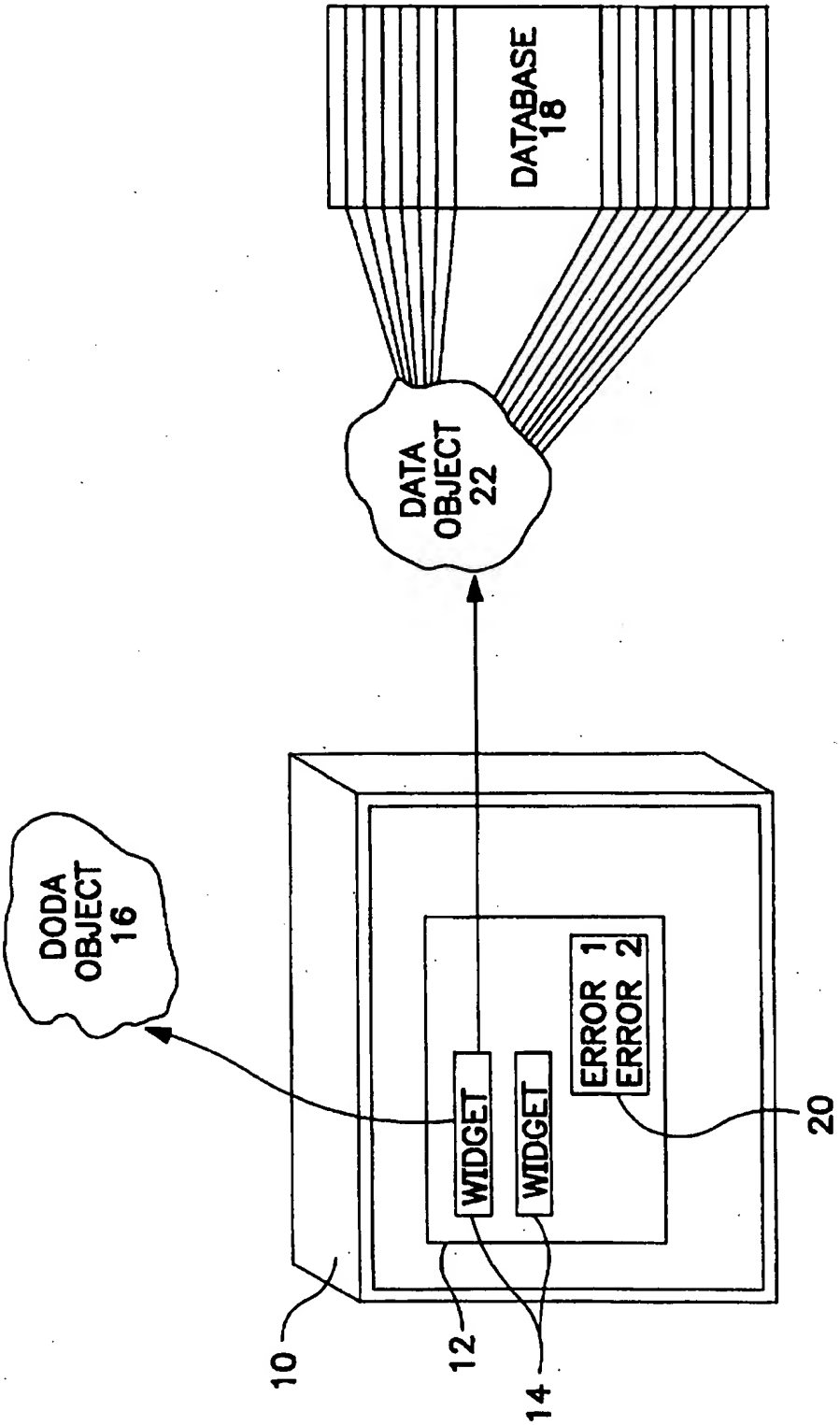

DATABASE 18
DATA OBJECT 22
DODA OBJECT 16
WIDGET
WIDGET
ERROR 1
ERROR 2
10
12
14
20

FIG. 1

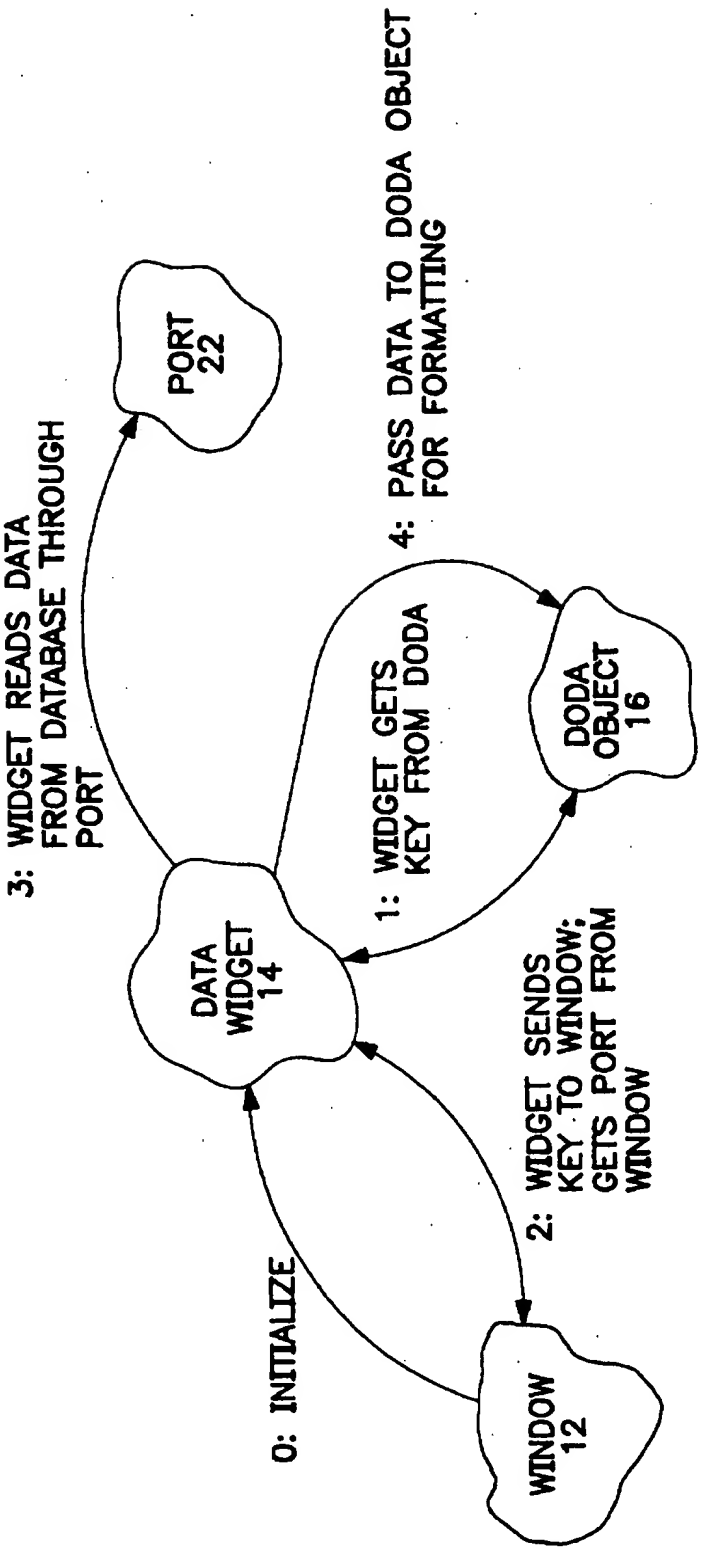

PORT
22
4: PASS DATA TO DODA OBJECT
FOR FORMATTING
3: WIDGET READS DATA
FROM DATABASE THROUGH
PORT
DODA
OBJECT
16
1: WIDGET GETS
KEY FROM DODA
DATA
WIDGET
14
2: WIDGET SENDS
KEY TO WINDOW;
GETS PORT FROM
WINDOW
0: INITIALIZE
WINDOW
12

FIG. 2

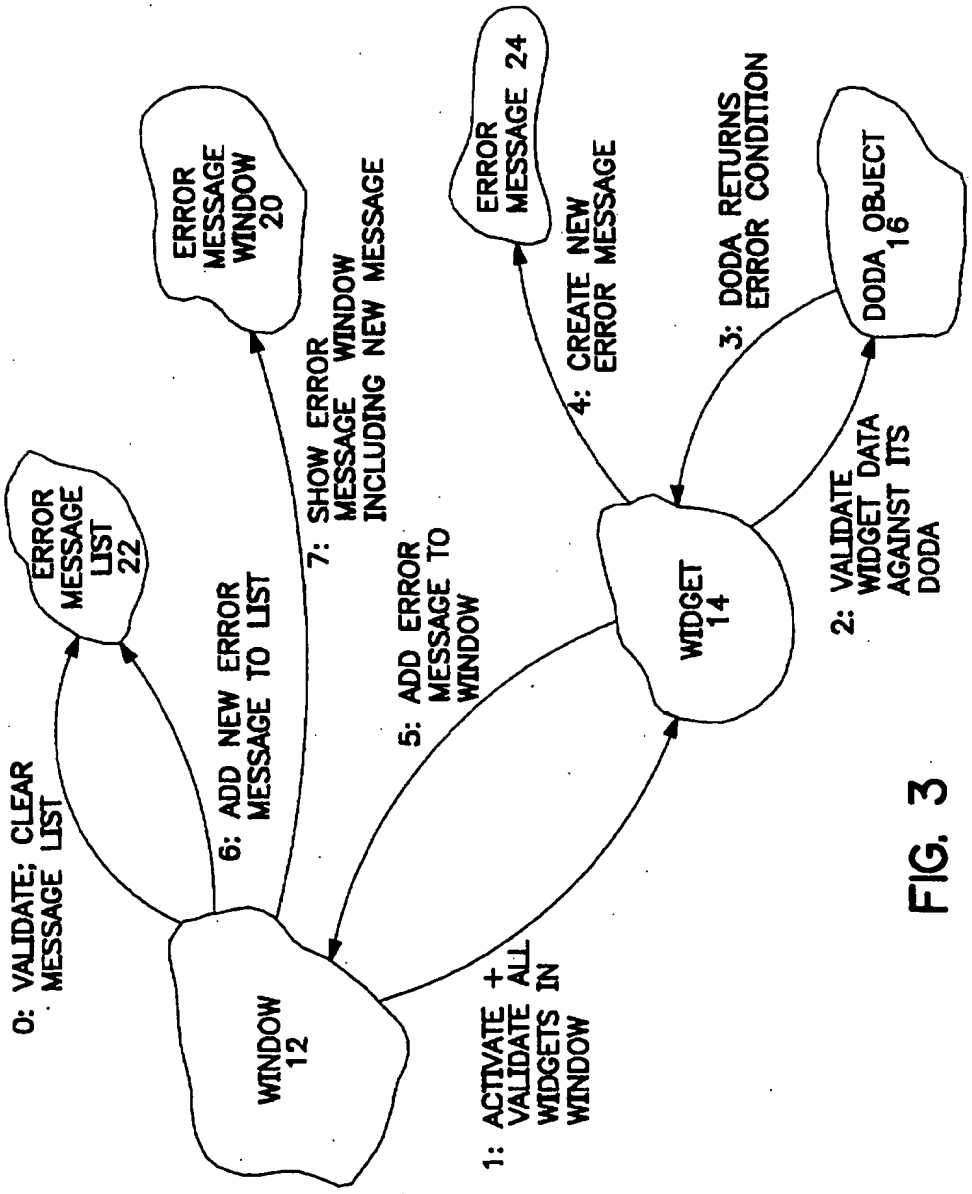

ERROR MESSAGE WINDOW 20
ERROR MESSAGE 24
DODA OBJECT 16
ERROR MESSAGE LIST 22
WIDGET 14
WINDOW 12
0: VALIDATE; CLEAR MESSAGE LIST
6: ADD NEW ERROR MESSAGE TO LIST
7: SHOW ERROR MESSAGE WINDOW INCLUDING NEW MESSAGE
5: ADD ERROR MESSAGE TO WINDOW
4: CREATE NEW ERROR MESSAGE
3: DODA RETURNS ERROR CONDITION
2: VALIDATE WIDGET DATA AGAINST ITS DODA
1: ACTIVATE + VALIDATE ALL WIDGETS IN WINDOW

FIG. 3

METHOD AND APPARATUS FOR VALIDATING DATA ENTERED BY A USER

BACKGROUND

This invention relates to the validation of data entered into a computer data processing system.

Many types of applications involve the manual entry by a human user of various types of information into a computer database for storage or processing. One type of system involves a generic library to which many different developers have access. One example of such a system is one which is used to store all of the files for a business organization, such as its technical and financial data. Any employee of the business may retrieve and process the information stored in the system.

In such applications, ensuring that the information entered into the database is accurate is an important factor, since very often the users who retrieve and process the information stored in the database are not the users who originally entered the data into the database.

SUMMARY OF THE INVENTION

The present invention relates to a window validation system and apparatus for verifying that data entered into a computer data processing system is correctly formatted. The system includes a plurality of windows, each including at least one widget area into which data is entered by the user. A "doda object" is associated with each widget and contains keys identifying where the data is to be stored in a database, as well as member functions which operate to format the entered data for storage and format information retrieved from the database for display in the widget. During the validation procedure, all of the data associated with each of the widgets in a window are first examined for errors in conjunction with its associated doda object. In addition, a second validation step is performed in which the data associated with certain widgets are compared to those of other widgets in the window, in accordance with an algorithm selected by the developer, to detect the presence of errors resulting from data entered inconsistently between widgets. If any errors are detected, an error message is displayed within a validation window. The widget in which the error was detected may then be selected by the user by double-clicking on the error message.

BRIEF DESCRIPTION OF THE DRAWING FIGURES

FIG. 1 illustrates the apparatus of the present invention;

FIG. 2 illustrates the relationships between the various objects; and

FIG. 3 is an object diagram illustrating the validation apparatus and procedure.

DETAILED DESCRIPTION

Some familiarity with the basic aspects of a computer data processing system is assumed in the following discussion. Accordingly, all of the hardware portions of the basic data processing system will not be illustrated. Briefly, with reference to FIG. 1, the present apparatus generally comprises a CPU having a database, a display area 10 such as a CRT display, and an apparatus such as a keyboard and/or a mouse or trackball for entering data into the system.

The visual display or screen 10 includes at least one window area 12. The term "window" will be used hereinafter in a manner known in the art to denote a rectangular portion of the display area which contains alphanumeric or graphic symbols representing various programs or operating states for the system. More than one window may be displayed on the screen in an overlapping fashion, to represent the several application programs or operating states available to the user, with the topmost window generally representing the currently active window. The user may select any of the available windows to represent the active window by changing the "focus" of the screen onto that window.

The visual display includes a cursor or arrow (hereinafter called a cursor) which can be moved around the display area with the aid of a mouse, trackball, or keys on the keyboard. The cursor can be used to select and activate various applications represented by graphic symbols on the display area. For example, the user can "re-focus" onto another window by moving the cursor into that window area and pressing a key or button when the cursor is in the desired location (called "clicking" on the desired location, or more commonly, double-clicking by pressing a key twice in rapid succession), in a manner known in the art. In this manner, the user can alternate between entering data in different windows.

Additionally, each window includes graphic symbols, generally known and "icons", each of which represents a program or application available to be executed. The user can select one of these icons, and initiate execution of the application or program represented y the icon, by clicking on it in the display area, in a manner known in the art.

In the present case, with reference to FIG. 1, each window includes at least one symbol called a widget 14. Each widget 14 within the window 12 is a subset of the display area into which data that is to be stored in the database or otherwise processed by the system, such as a text field, may be entered via the keyboard. Each window may simultaneously include many widgets, so that a user can enter and manipulate many different types of data. In general, user may select one of many widgets displayed in a window by moving the screen cursor with the aid of a mouse or trackball into the widget area, and pressing a button on the mouse or keyboard to make the desired widget into the "active" widget, and then keying in the appropriate data.

By way of example, a system may be programmed to operate a computer spreadsheet program containing many fields for entering financial data. A single data field or group of related data fields may be considered a widget into which particular data may be typed and stored in the database for later use.

Each widget is operatively linked to the database 18 by an interface called a "data object" or "port", illustrated in FIG. 1 as element 22. The data object 22 utilizes the keys obtained from a doda object 16 (discussed below) to access the appropriate location or address in the database at which the desired widget data has been or is to be stored.

Each widget 14 is associated with a "doda" (acronym for "data object data attribute") object, shown symbolically in FIG. 1 as element 16. The doda object 16 maintains (usually as a set of flags, pointers, and the like) information or "keys" which identify the proper address location(s) in the dataport 22 at which the converted data is to be stored, or from which data is to be retrieved and transferred to the widget for display.

In addition, in typical data processing systems the data entered into or displayed in a widget 14 is usually in a different format than the information symbols that are actu-

ally stored in the database **18** as representing that data. By way of example, the raw alphanumeric data from the widget **14** may be subjected to various compression techniques prior to storage in the database **18** in order to minimize the amount of memory space occupied by the data, or it may be grouped or packeted with other data prior to storage in order to maximize storage efficiency, or the data may be processed according to any number of other procedures known in the art. In one common arrangement, data which is displayed in a widget **14** may be considered to have been "formatted" for display, as for example, when a string of numbers representing a calendar date is placed in the well-known MM/DD/YY format, while the same data may be stored in the database **18** without the intervening slash marks or hyphens, in what may be called an "unformatted" manner. Although all of these different examples will fall within the scope of this invention, the following discussion will adopt, for illustrative purposes, the terminology "formatted" to refer to data that is in a format appropriate for display in a widget **14**, while data that is in a format that is appropriate for storage in the database **18** will be referred to as "unformatted" data.

In order to ensure that formatted data entered into a widget **14** is transformed into unformatted form for storage prior to conveying the data to the database **18**, and that unformatted data retrieved from the database **18** is properly formatted for display in a widget **14**, the doda object **16** contains information and algorithms (also called "member functions") which identify the data format corresponding both to the database **18** and the widget **14** to which the doda object **16** is linked, and operates to transform the data into the proper format for the desired operation (storage or display). In other words, any formatted data sent from a widget **14** to the database **18** is first transformed by the doda object **16** for that widget into unformatted data for storage. Similarly, any unformatted data retrieved from the database **18** into a widget **14** is formatted for display by the doda object **16**.

As will be evident, different doda objects will include different member functions or algorithms depending upon the type of data that is to be displayed in their associated widgets; for example, one widget may be configured to show a calendar date, while another may be configured to display a name, an address, a dollar amount, or any other type of information. The particular member functions used by each doda object are not limited and may be selected or programmed by the developer.

It should be noted that where a data is entered into a system one character at a time, as through a keyboard, the data may also be considered to be unformatted. Thus, as data is being entered in this manner, it may also be operated upon by the appropriate member functions to place it into the correct format for the widget.

Additionally, a single data processing system may include multiple databases which may be accessed during different program sequences. A particular widget may therefore be linked to different data objects or ports when different program sequences are being executed. In this situation, the program window **12** may be configured to contain information (pointers) identifying which data ports are linked with which widgets. In this case, any keys retrieved from the doda object will first be passed to the window, which will locate the data port to be accessed with that key, and then send the key to the appropriate port for further processing.

Both the doda object **16** and the data object **22** are hidden from the user, who sees only the widget data on his CRT display **10**.

The window **12** includes, in addition to the displayed widgets **14**, a smaller error validation window **20** which is used to store and display any error messages that may occur during operation. The validation window **20** receives error messages as strings, along with optional pointers identifying the source or location of the error and a severity indicator, and displays the strings in the validation window.

It is a fundamental characteristic of this type of system that the widget **14** does not, by itself, "how" or include information indicating the form of the data that is being manipulated. Instead, all information describing what characters are allowed to be entered, the format of the data, etc., are known to and processed by the doda objects.

Within this system, validation of data entered within a window may be performed as described below. The validation procedure may be initiated at the request of the user, as by entering a particular command or pressing a particular key, or automatically, for example, at certain time intervals, or in response to the occurrence of certain events, such as, for example, when the user indicates that he has completed data entry. Once initiated, the validation process is performed on all widgets within a particular window (usually the active window, although other windows may be selected as desired).

During window validation, the data entered into each widget **14** within the window **12** is verified by the system in conjunction with the associated doda object **16** using any number of alternative error-checking algorithms as desired. For example, the doda object **16** may be instructed to check if the entered data contains the correct number and type of text characters. Additionally, the system may instruct the doda object **16** to compare the data entered in the widget against a database of all valid information that can be entered into that widget; for example, a calendar date entry may be compared against a list of all valid calendar dates to ascertain that an entry of "December 32" is not a valid entry. Numerous other algorithms may be used, either alone or in combination with other algorithms. The data entered in each widget within the desired window is checked in this manner by the associated doda object.

In the event that invalid data is discovered to have been entered into a widget, the system concludes that an error condition exists, and generates an appropriate error message string for display in the validation window **20**. This string includes an alphanumeric text message designed to inform the user of the presence of an error. The error message also includes a pointer identifying the widget in which the error is deemed to have occurred.

When the error message(s) have been displayed in the validation window **20**, the user may use the mouse to "click" on any of the displayed error messages, causing screen focus to shift to the widget indicated by the pointer associated with that error message string, to enable the user to re-enter the correct data into the widget. After the data is re-entered by the user, validation is repeated to ensure that no additional errors exist.

Additionally, the window performs a second validation step of comparing the data entered into different widgets within the window, according to any of several possible algorithms designed by the developer, in order to detect errors resulting from the entry of inconsistent or incompatible data in different widgets. For example, if the program being used requests the entry of a beginning date of a project or occurrence into one widget and an end date into another widget, the window may include an algorithm which compares the two to ensure that the end date is later in time than

the beginning date, and results in an error if it is not. Similarly, in a spreadsheet program, the window could be designed to check that all of the figures in a particular column added up to the correct total or are greater than figures in another column. Any combination of algorithms could be utilized, depending upon the particular application to which the window is directed and the types of errors that the user desires to prevent.

In the event that an error is detected, the window generates an error message string and displays it in the validation window **20**. The string may also include a pointer identifying the particular widget(s) whose data created the error. If the user subsequently clicks on one of the error messages in the validation window **20**, then screen focus could be shifted to that widget, or to any other portion of the window as designed by the developer. If the data from more than one widget contributed to the error, then focus could be shifted to an appropriate default widget.

If no error condition is detected in either step of the validation process, then the window proceeds according to its originally programmed sequence.

In order to further conceptually illustrate the above process, the following summary of one example of a possible program sequence is presented. In response to a particular sequence, a widget **14** retrieves a key from its doda object **16** and passes it to the window **12** with a request that the window identify the data port **22** that the key references. When the widget receives the port identifier, it passes the key to the port **22**. With the aid of the key, unformatted data is retrieved from the database through the data object or port **22** and passed to the widget **14**. The widget **14** then passes the data to the doda object **16** which formats the data according to its member functions and returns the formatted string to the widget **14** for display. Once displayed, a user may choose to modify or update the data by manipulating the cursor and typing in new characters into the widget. The newly entered characters are passed by the widget **14** to the doda object **16**, which either accepts the character, rejects the character, or adds format characters (e.g., adds a hyphen to a phone number), as appropriate. When the widget **14** is to validate its data, either automatically or in response to a separate instruction, its data is passed again to the doda object **16** and checked against various error checking algorithms and member functions. When the widget data is to be written to the database **18**, the key is again retrieved from the doda object **16** and sent to the window **12** to obtain the appropriate port **22**, and the data (properly formatted by the doda object for storage) is transferred to the port and ultimately to the database **18** for storage.

FIGS. **2** and **3** illustrate the present invention diagrammatically, by object scenario diagrams which show the various steps in the sequence. FIG. **2** depicts an operation in which data is retrieved from the database for display in a widget, and FIG. **3** depicts a validation operation.

Referring first to FIG. **2**, some of the various objects of the present invention are shown, including a window **12**, a data widget **14** within the window **12**, a doda object **16** associated with the widget **14**, and a port **22** linking the widget **14** to a database (not shown in FIG. **2**). Each of the arrows connecting the objects of FIG. **2** represents a step of the sequence, with the arrows indicating the direction of data or information flow, from one object to another object. Each arrow is numbered and labeled to describe the represented function. Thus, the steps illustrated in FIG. **2** to retrieve data from the database for display in a widget are as follows:

0. Initialize the window and all widgets within the window

1. The widget **14** retrieves the key corresponding to the desired data from its doda object **16**.

2. The widget **14** sends the key to the window, along with a request for identification of the appropriate port **22** through which the data may be retrieved; the window responds by sending a pointer identifying the port **22**.

3. The widget **14** reads the appropriate data from the location in the dataport **22** corresponding to the key.

4. The retrieved data is passed to the doda object **16** for formatting.

When all of these steps have been completed, the data is ready for display in the widget **14**

FIG. **3** shows many of the same objects as in FIG. **2**, with the addition of new objects corresponding to the error message window **20**, an error message list **22**, and an error message **24**. The steps in FIG. **3** illustrate the validation procedure, as follows:

0. Validate the window **12** and clear the error message list **22** of any old messages that might have been retained from the last validation procedure.

1. Activate and validate all widgets **14** in the window **12**.

2. Validate the data entered into each widget **14** against the information (error checking algorithms, member functions, flags, pointers, etc.) in its doda object **16**.

3. If the doda object **16** indicates that no error conditions exist, proceed in accordance with the original program sequence. Otherwise, the doda object **16** will return an indication of an error condition.

4. In response to the detection of an error condition, the widget **14** creates a new error message string **24** which describes the error. The widget also creates a pointer (not shown) which identifies the widget.

5. The error message **24** and any accompanying pointers are sent to the window **12**.

6. The error message **24** is added to the error message list **22**.

7. The error message string **24** is displayed in the error message window **20**. If the user then double clicks on the error message in the window, focus is shifted to the widget **14** in which the error was detected.

Steps **2–7** are repeated for each active widget **14** in the window **12**.

Thus, the present method and apparatus provides for validation of data entered into one or more widgets **14** within a window **12**. In the event an error condition is detected, the user can easily locate the source of the error by clicking on the appropriate error message within the validation window **20**, which shifts the screen focus to the widget containing the erroneous entry. The user can then re-enter the data. After re-entry, the validation process is repeated, to ensure that the new dam also does not contain an error. The process continues until the data in all of the widgets within the window has been validated.

Although the present invention has been described hereinabove with particular reference to the preferred embodiment, it should be understood that various alternatives, modifications and adaptations may be made without departing from the scope of the invention as set forth in the appended claims.

We claim:

1. A system for storing and retrieving information to and from a database, comprising:

keyboard means for entering data into the system;

display means, for visually displaying alphanumeric and graphic symbols representing requests for data and the data entered into the system, said display means comprising

at least one window disposed within the display means,

a validation window associated with each window and capable of being displayed on said display means within said window, and

at least one widget disposed within each of said windows, wherein all data entered into the system via said keyboard means is associated with at least one widget;

means for detecting an error condition relating to the data associated with each of said widgets within a window;

means responsive to said means for detecting an error condition for creating an error message string for display in the validation window associated with the window in which said error condition was detected, said means also creating pointer data identifying the widget in which the error condition was detected.

2. A system for storing and retrieving information to and from a database according to claim 1, further comprising re-entry means responsive to said means for creating an error message, said re-entry means re-displaying the widget identified by the pointer data associated with the error message string so as to enable re-entry of the data associated with that widget.

3. A data processing system having the capability of validating data entered into the system, comprising:

display means, comprising

at least one window displaying a plurality of requests for information, and

at least one widget within each window associated with at least one of said displayed requests for information;

means for entering data into the system such that the entered data is associated with at least one of said widgets;

a doda object associated with each of said widgets, the doda object including member functions adapted to ascertain whether data entered in association with the widget is correctly formatted;

means for detecting an error condition in the data objects associated with each of said widgets, based upon the doda object associated with that widget;

means responsive to the detection of an error condition for displaying an error message on a portion of said display means and for pointing to the location of the error condition.

4. A data processing system having the capability of validating data entered into the system according to claim 3, further comprising re-entry means responsive to said means for displaying an error message, said re-entry means re-displaying the location of the error condition so as to enable re-entry of data.

5. A method of validating information entered into a data processing system having visual display means, comprising the steps of:

creating at least one window area on said visual display means;

displaying a plurality of requests for information in said window area;

displaying a plurality of widgets in said window area, each widget associated with at least one of said requests for information;

associating data entered in response to said requests for information with at least one of said widgets;

creating at least one doda object associated with each of said widgets, said doda object including member functions adapted to ascertain whether data associated with the widget is properly formatted;

for each widget, detecting, with the assistance of the doda object for that widget, the presence or absence of an error condition in the data associated with that widget, and in response to the detection of an error condition, creating an error message string and a pointer identifying the widget associated with the data object in which the error condition was detected;

displaying for selection by a system operator, all error message strings on a portion of said window area;

in response to the selection by a system operator of one of the displayed error message strings, re-displaying the widget identified by the pointer associated with the selected error message string so as to enable re-entry of the data associated with that widget.

6. A method of validating information entered into a data processing system having visual display means, comprising the steps of:

creating at least one window area on said visual display means;

displaying a plurality of requests for information in said window area;

displaying a plurality of widgets in said window area, each widget associated with at least one of said requests for information;

as data is entered into the system, associating the entered data with at least one of the widgets;

creating at least one doda object associated with each of said widgets, said doda object including member functions adapted to ascertain whether entered data associated with the widget is properly formatted;

for each widget, detecting, with the assistance of the doda object for that widget, the presence or absence of an error condition within the data associated with that widget by comparing said data with a stored database of available valid data, and in response to the detection of an error condition, creating an error message string and a pointer identifying the widget associated with the data in which the error condition was detected;

for each widget, detecting, with the assistance of the doda object for that widget, the presence or absence of an error condition within the data associated with that widget by comparing said data with the data associated with other widgets in said window area, and in response to the detection of an error condition, creating an error message string and a pointer identifying the widget associated with the data in which the error condition was detected;

displaying for selection by a system operator, all error message strings on a portion of said window area;

in response to the selection by a system operator of one of the displayed error message strings, re-displaying the widget identified by the pointer associated with the selected error message string so as to enable re-entry of the data associated with that widget.

* * * * *

US005987469A

United States Patent [19]

Lewis et al.

[11] Patent Number: 5,987,469

[45] Date of Patent: Nov. 16, 1999

[54] METHOD AND APPARATUS FOR GRAPHICALLY REPRESENTING INFORMATION STORED IN ELECTRONIC MEDIA

[75] Inventors: James D. Lewis, Wyckoff, N.J.; Bogdan N. Reznik, Brooklyn, N.Y.; Eran Librach, Fair Lawn, N.J.; Bryce H. Bonnet, New Milford, N.J.; Nancy J. Lewis, Wyckoff, N.J.

[73] Assignee: Micro Logic Corp., South Hackensack, N.J.

[21] Appl. No.: 08/855,553

[22] Filed: May 13, 1997

Related U.S. Application Data

[60] Provisional application No. 60/017,400, May 14, 1996.

[51] Int. Cl.[6] G06F 9/00

[52] U.S. Cl. 707/102; 707/104

[58] Field of Search 707/100, 200, 707/104

[56] References Cited

U.S. PATENT DOCUMENTS

5,058,144	10/1991	Fiala et al.	375/240
5,551,022	8/1996	Tariq et al.	707/104
5,606,669	2/1997	Bertin et al.	395/200.53
5,644,763	7/1997	Roy	707/101
5,724,576	3/1998	Letourneau	707/100

OTHER PUBLICATIONS

Microsoft® Windows™ For Workgroups User's Guide, p. 111, Copyright 1985–1993.

WinSurfer™ and tree Viz™ program webpages.

Primary Examiner—Paul R. Lintz

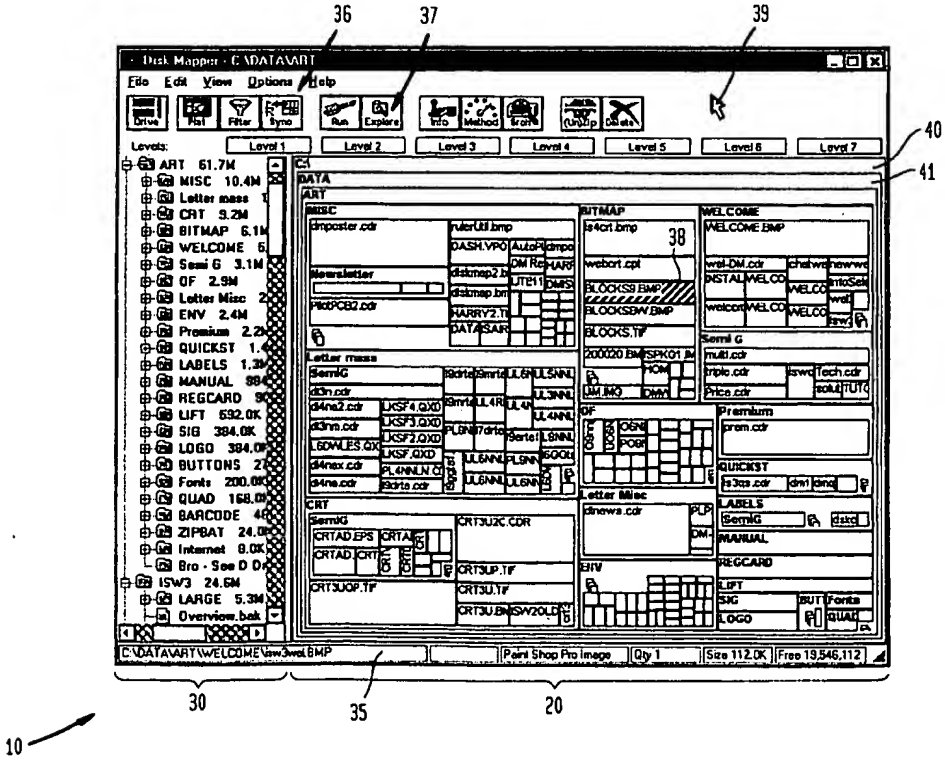
Assistant Examiner—John Gladstone Mills, III

Attorney, Agent, or Firm—Lerner, David, Littenberg, Krumholz & Mentlik, LLP

[57] ABSTRACT

A method and apparatus is provided for displaying nested rectangles which graphically illustrate the directories and files located in a storage medium such a computer hard disk drive or the nodes of a tree data structure. The sizes of the rectangles are proportional to the size of the directory or file they represent, and the rectangles are nested in a recursive manner in accordance with the branch structure of the directories and files on the storage media. Preferably, the dimensions of the rectangles are chosen to maximize the number of rectangles, especially those capable of displaying the names of the directories and files. Yet further, the invention preferably assigns colors to the rectangles based upon certain criteria and suppresses directories or files from being shown if their rectangles will be too small to be accurately perceived.

26 Claims, 14 Drawing Sheets



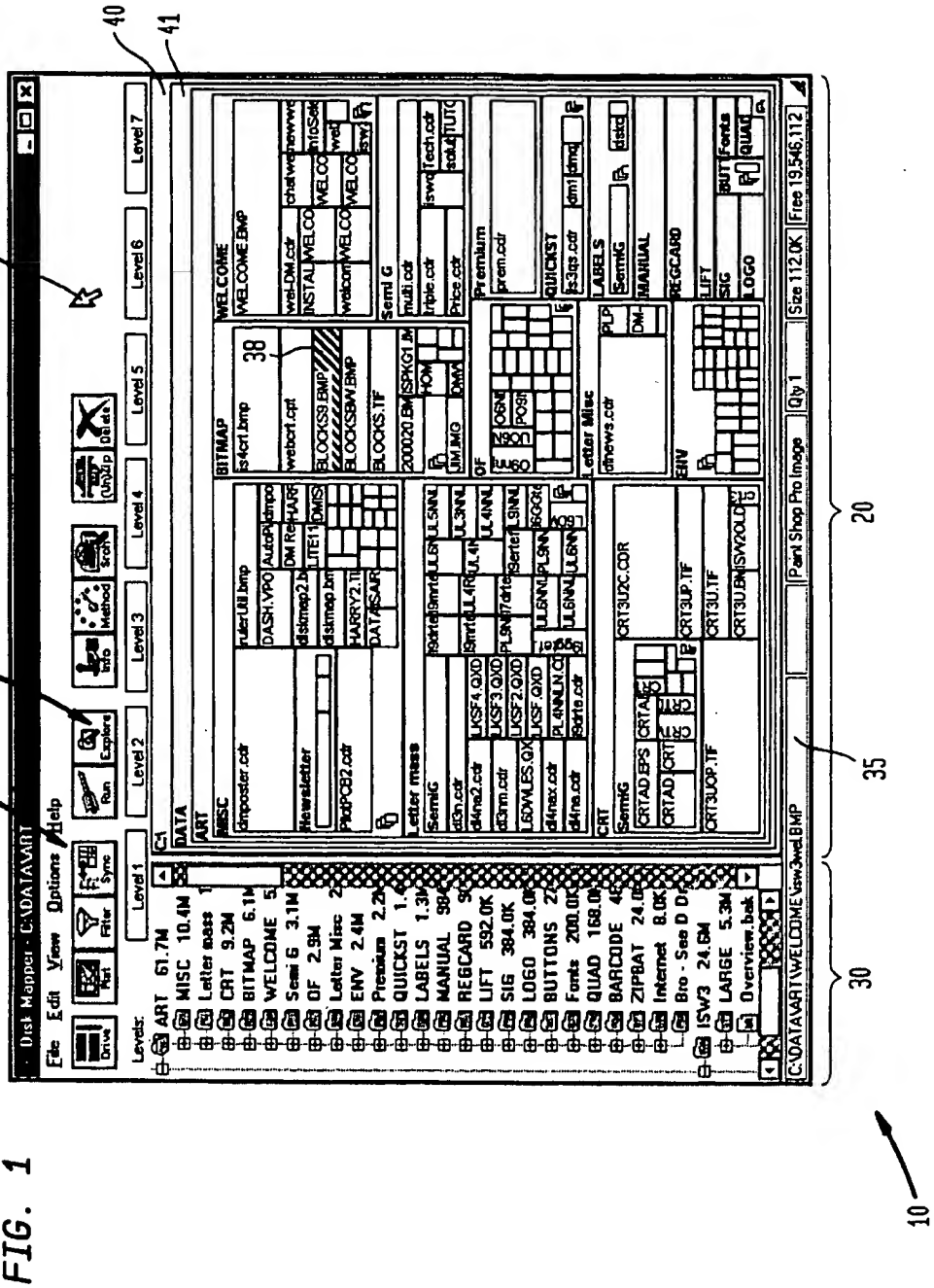

Disk Mapper - C:\DATA\ART
File Edit View Options Help
Drive Run Explorer Info Method ScanN UnZip Delete
Levels: Level 1 Level 2 Level 3 Level 4 Level 5 Level 6 Level 7
C:\DATA\ART\WELCOME\swJwel.BMP
Paint Shop Pro Image
Qty 1
Size 112.0K
Free 19,546,112
36
37
39
40
41
38
30
35
20
10

FIG. 1

FIG. 2

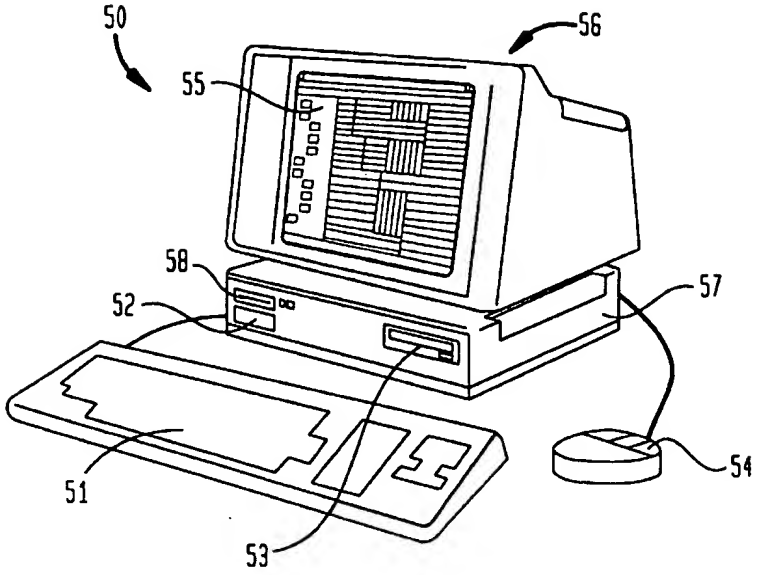

50
56
55
58
52
57
54
51
53

FIG. 3

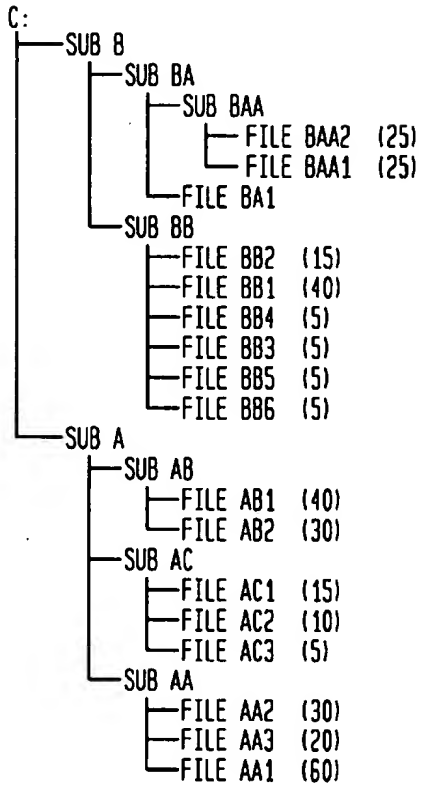

C:
SUB B
SUB BA
SUB BAA
FILE BAA2 (25)
FILE BAA1 (25)
FILE BA1
SUB BB
FILE BB2 (15)
FILE BB1 (40)
FILE BB4 (5)
FILE BB3 (5)
FILE BB5 (5)
FILE BB6 (5)
SUB A
SUB AB
FILE AB1 (40)
FILE AB2 (30)
SUB AC
FILE AC1 (15)
FILE AC2 (10)
FILE AC3 (5)
SUB AA
FILE AA2 (30)
FILE AA3 (20)
FILE AA1 (60)

FIG. 4

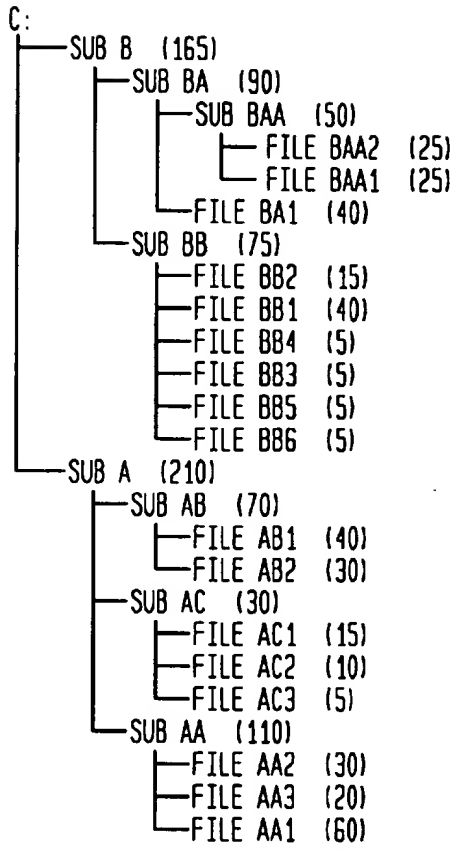

C:
SUB B (165)
SUB BA (90)
SUB BAA (50)
FILE BAA2 (25)
FILE BAA1 (25)
FILE BA1 (40)
SUB BB (75)
FILE BB2 (15)
FILE BB1 (40)
FILE BB4 (5)
FILE BB3 (5)
FILE BB5 (5)
FILE BB6 (5)
SUB A (210)
SUB AB (70)
FILE AB1 (40)
FILE AB2 (30)
SUB AC (30)
FILE AC1 (15)
FILE AC2 (10)
FILE AC3 (5)
SUB AA (110)
FILE AA2 (30)
FILE AA3 (20)
FILE AA1 (60)

FIG. 5

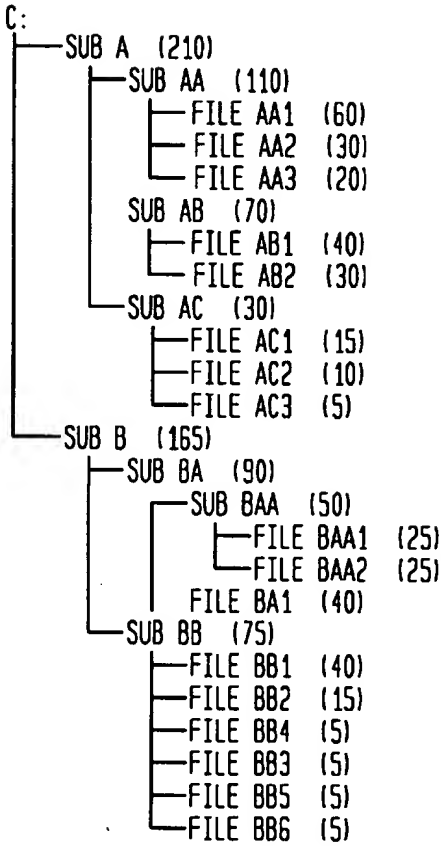

C:
SUB A (210)
SUB AA (110)
FILE AA1 (60)
FILE AA2 (30)
FILE AA3 (20)
SUB AB (70)
FILE AB1 (40)
FILE AB2 (30)
SUB AC (30)
FILE AC1 (15)
FILE AC2 (10)
FILE AC3 (5)
SUB B (165)
SUB BA (90)
SUB BAA (50)
FILE BAA1 (25)
FILE BAA2 (25)
FILE BA1 (40)
SUB BB (75)
FILE BB1 (40)
FILE BB2 (15)
FILE BB4 (5)
FILE BB3 (5)
FILE BB5 (5)
FILE BB6 (5)

FIG. 6

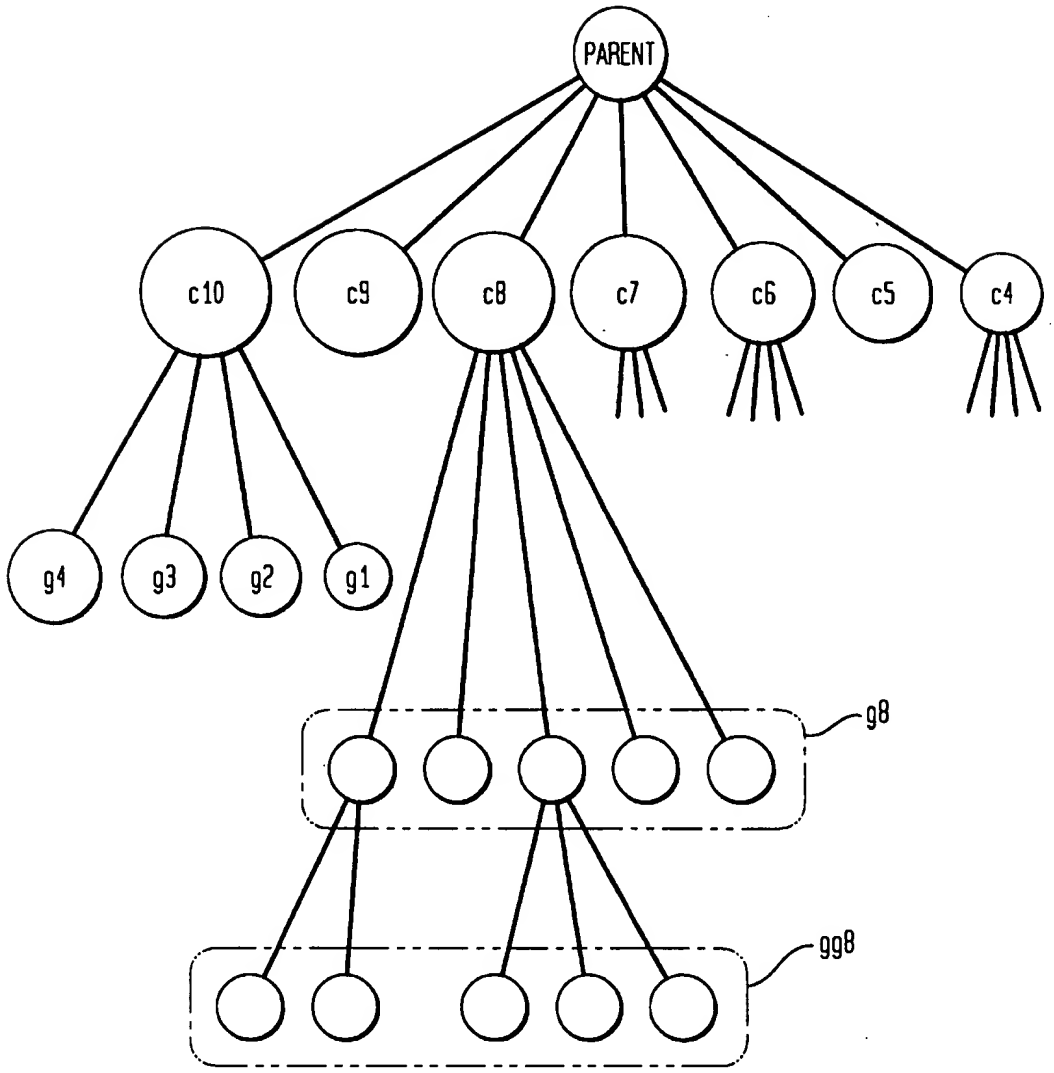


FIG. 7

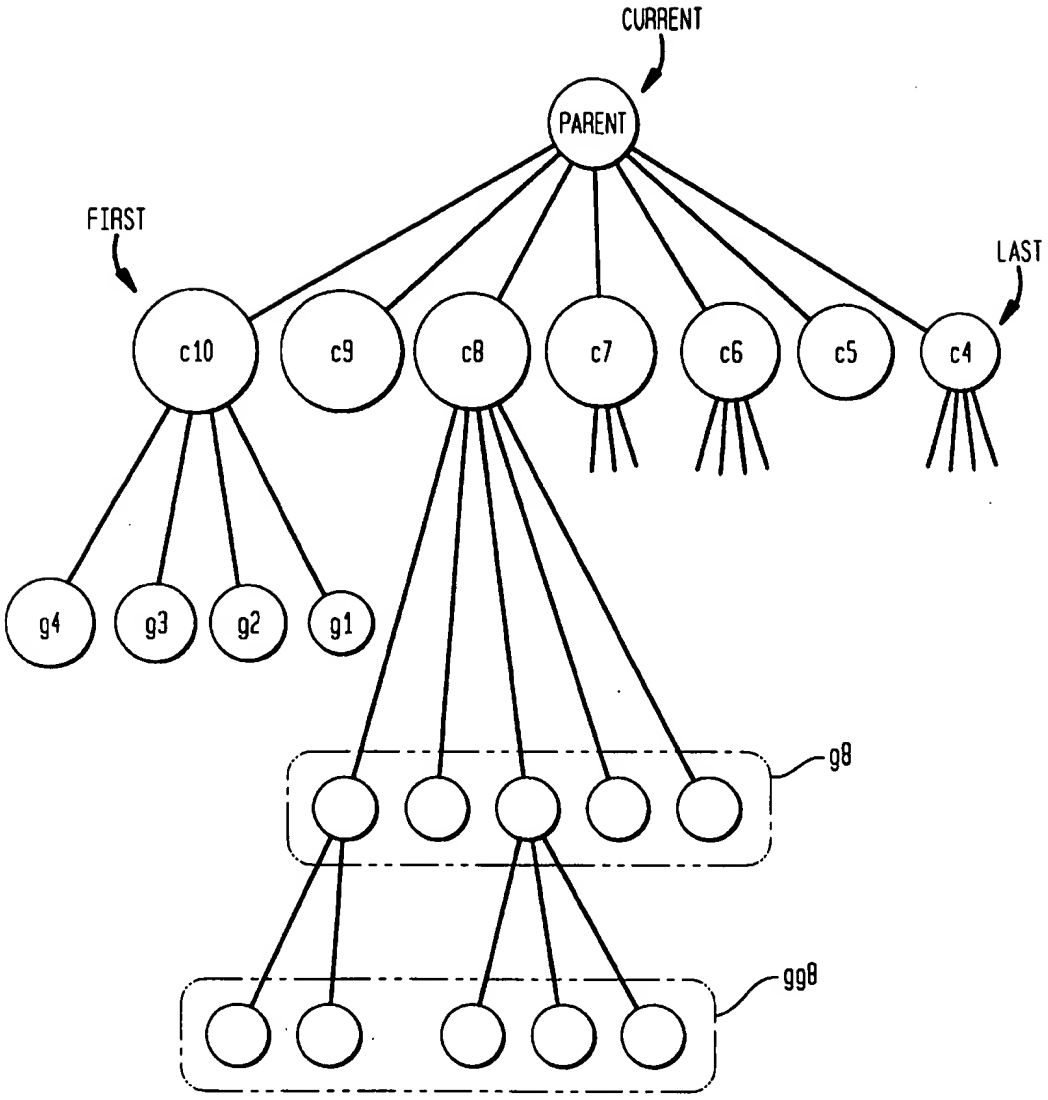

CURRENT
PARENT
FIRST
LAST
c10
c9
c8
c7
c6
c5
c4
g4
g3
g2
g1
g8
gg8

FIG. 8

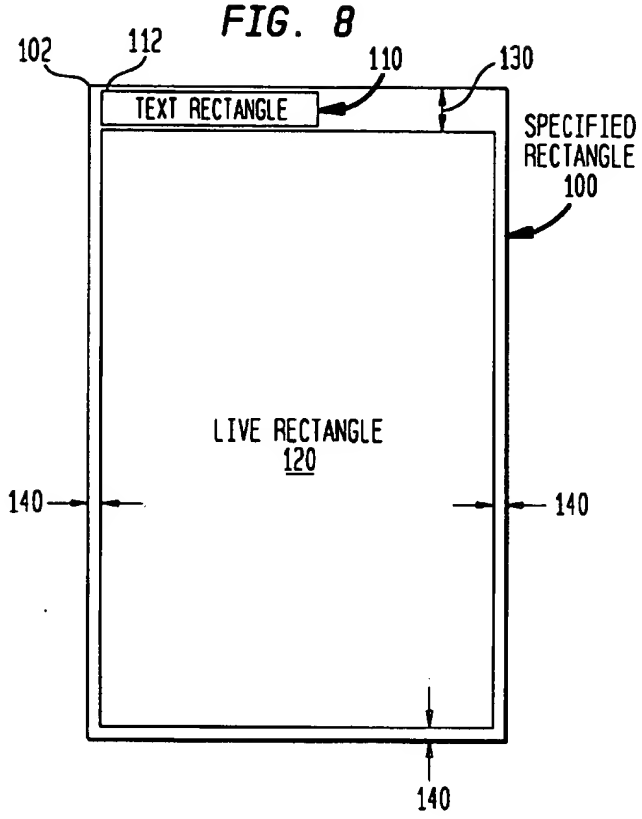

102
112
110
130
TEXT RECTANGLE
SPECIFIED RECTANGLE
100
LIVE RECTANGLE
120
140
140
140

FIG. 9

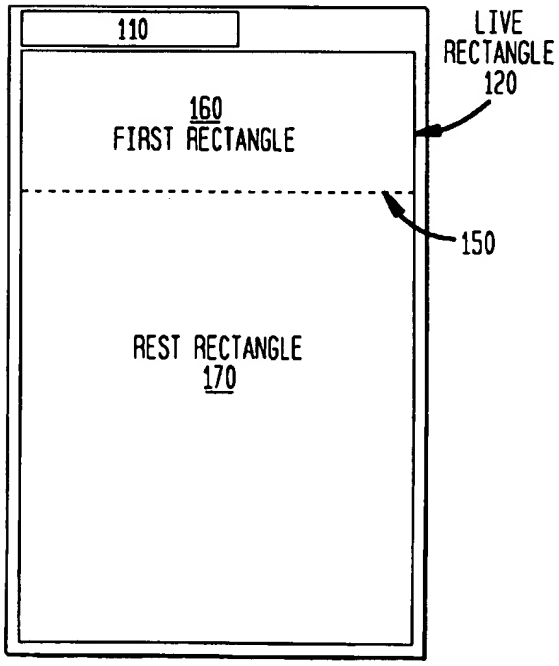

110
LIVE RECTANGLE
120
160
FIRST RECTANGLE
150
REST RECTANGLE
170

FIG. 10

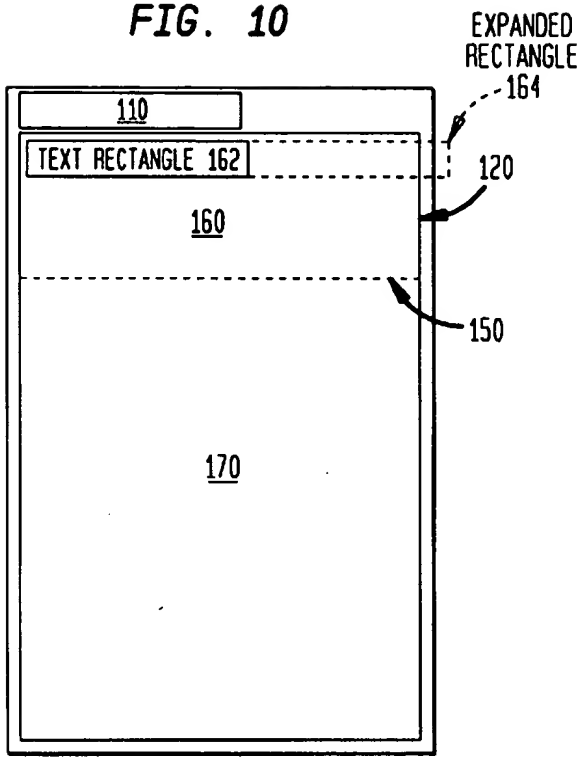

EXPANDED
RECTANGLE
164
110
TEXT RECTANGLE 162
120
160
150
170

FIG. 11

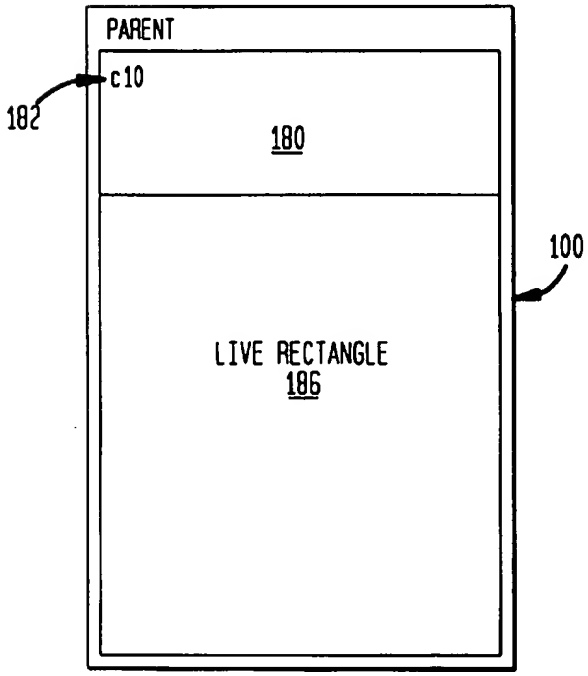

PARENT
c10
182
180
100
LIVE RECTANGLE
186

FIG. 12

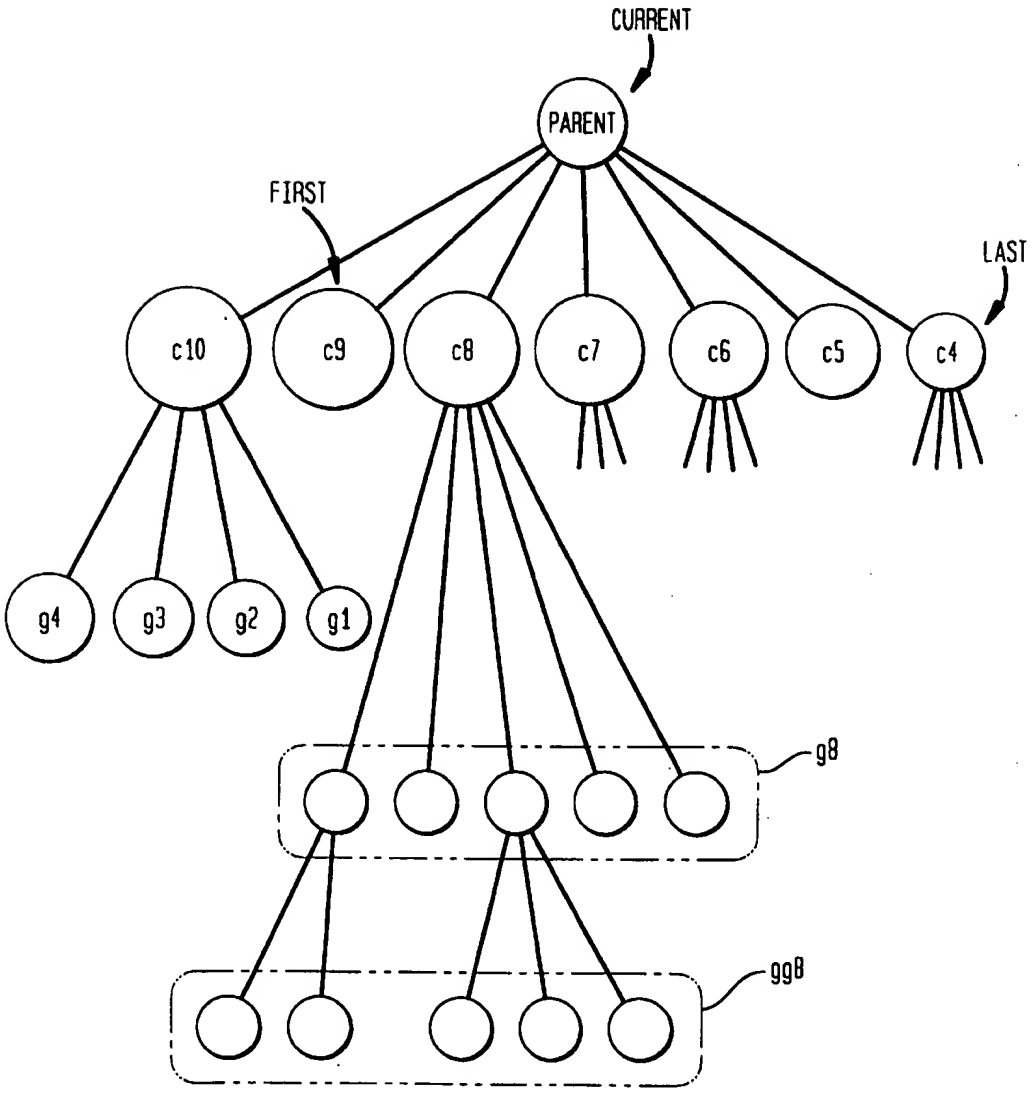

CURRENT
PARENT
FIRST
LAST
c10
c9
c8
c7
c6
c5
c4
g4
g3
g2
g1
g8
gg8

FIG. 13

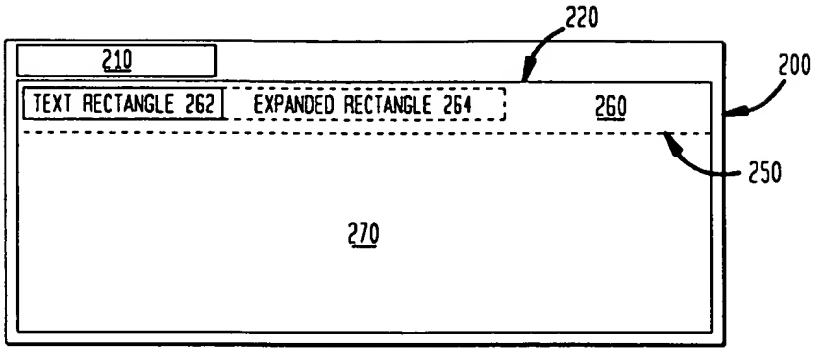

220
210
200
TEXT RECTANGLE 262
EXPANDED RECTANGLE 264
260
250
270

FIG. 14

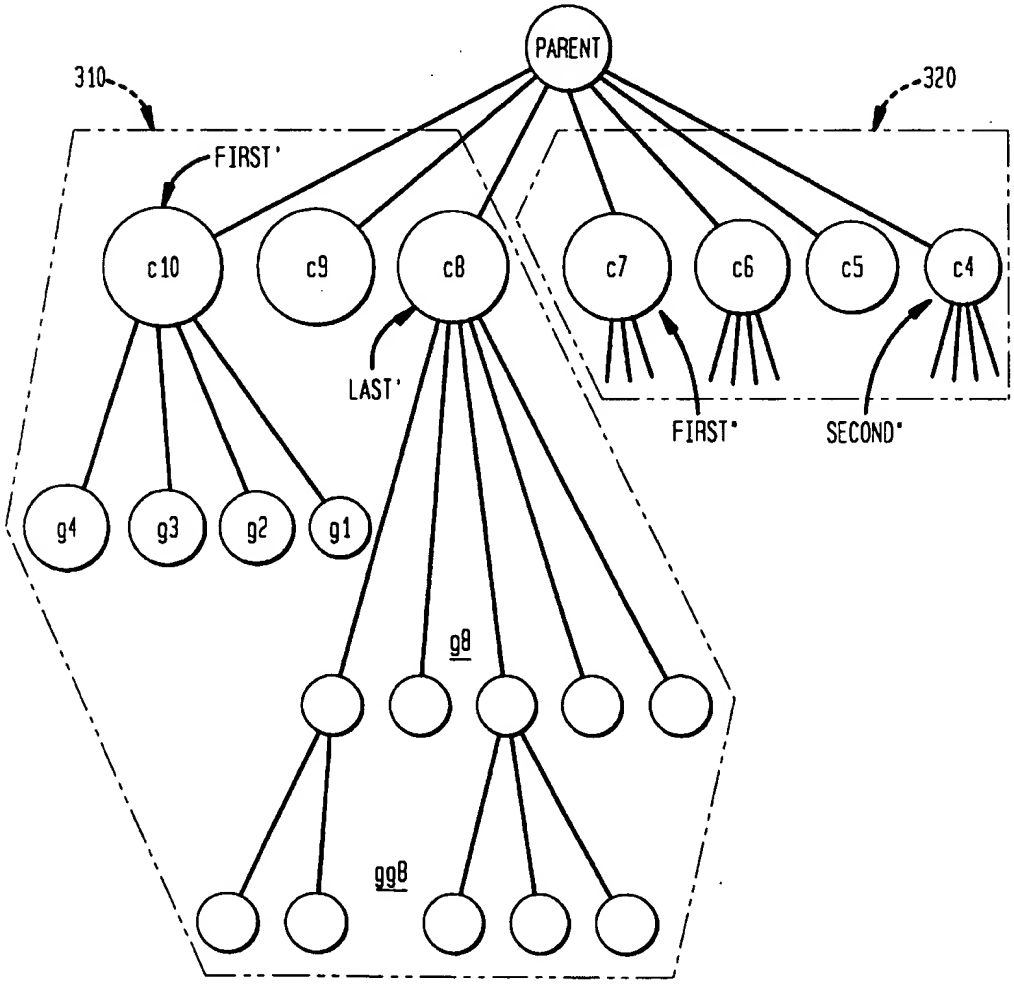

PARENT
310
320
FIRST'
c10
c9
c8
c7
c6
c5
c4
LAST'
FIRST'
SECOND'
g4
g3
g2
g1
g8
gg8

FIG. 15

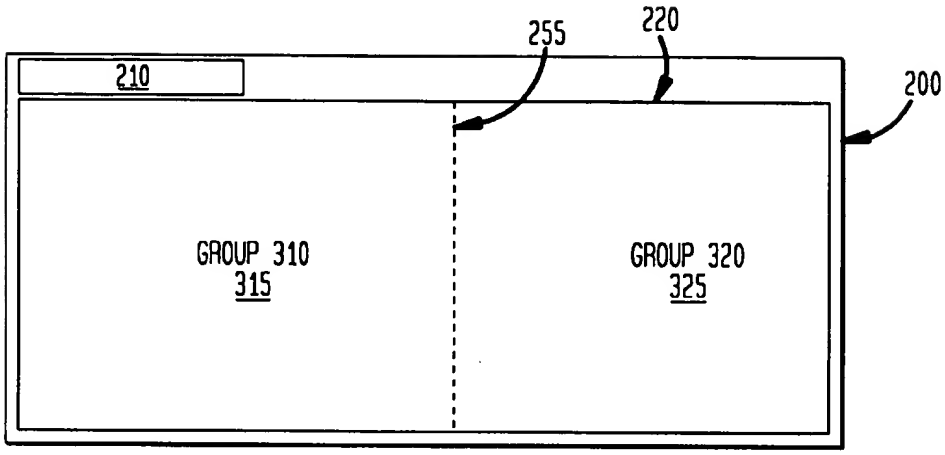

255
220
210
200
GROUP 310
315
GROUP 320
325

FIG. 16

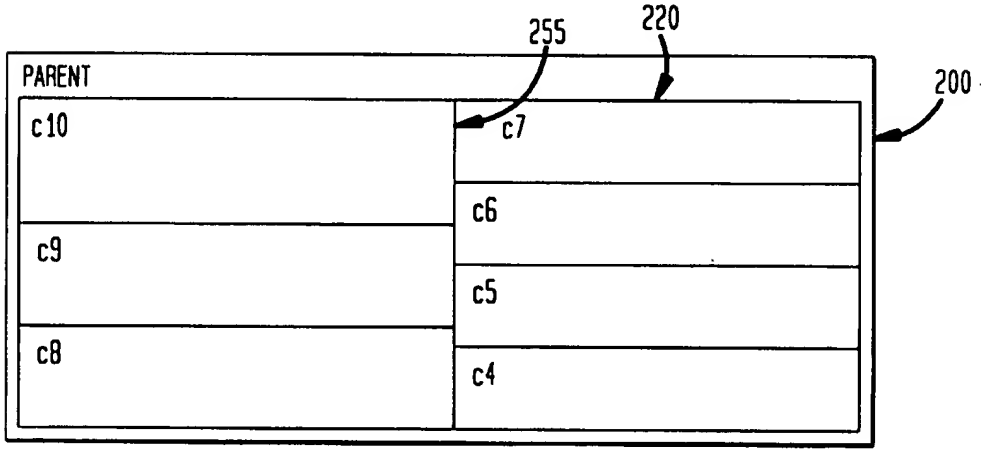

255
220
PARENT
200
c10
c7
c6
c9
c5
c8
c4

FIG. 17

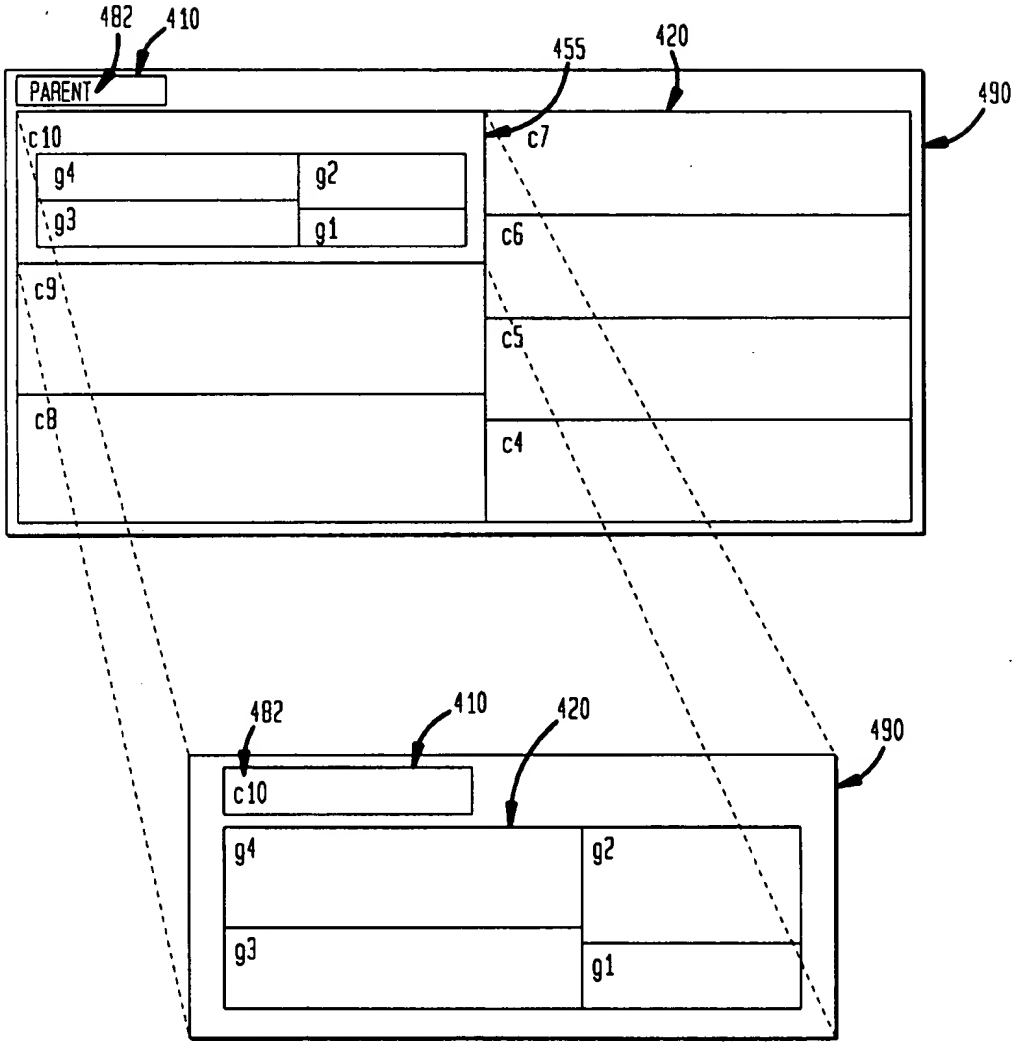

482
410
455
420
490
PARENT
c10
g4
g2
g3
g1
c7
c6
c9
c5
c8
c4
482
410
420
490
c10
g4
g2
g3
g1

FIG. 18

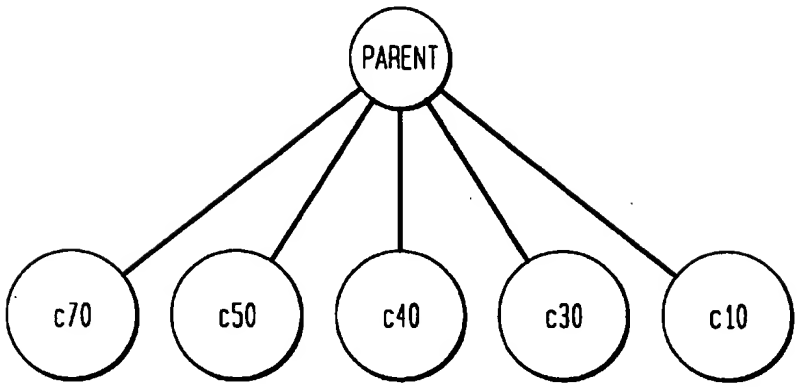

PARENT
c70
c50
c40
c30
c10

FIG. 19

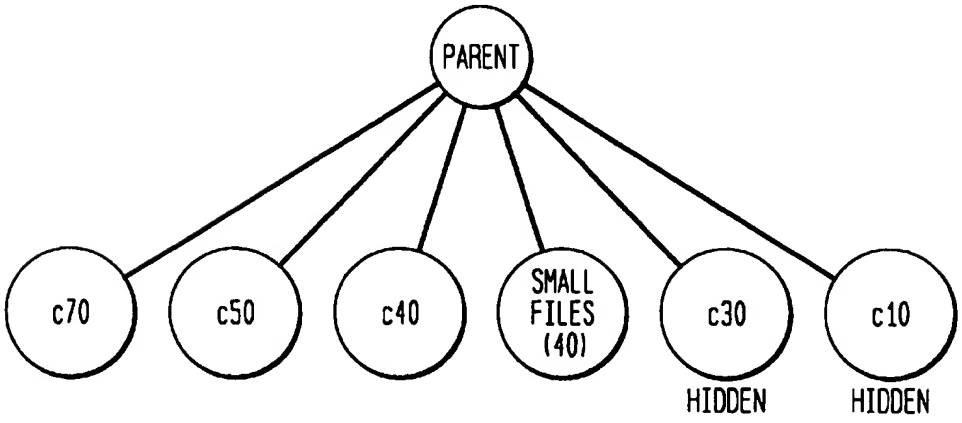

PARENT
c70
c50
c40
SMALL FILES (40)
c30
HIDDEN
c10
HIDDEN

FIG. 20

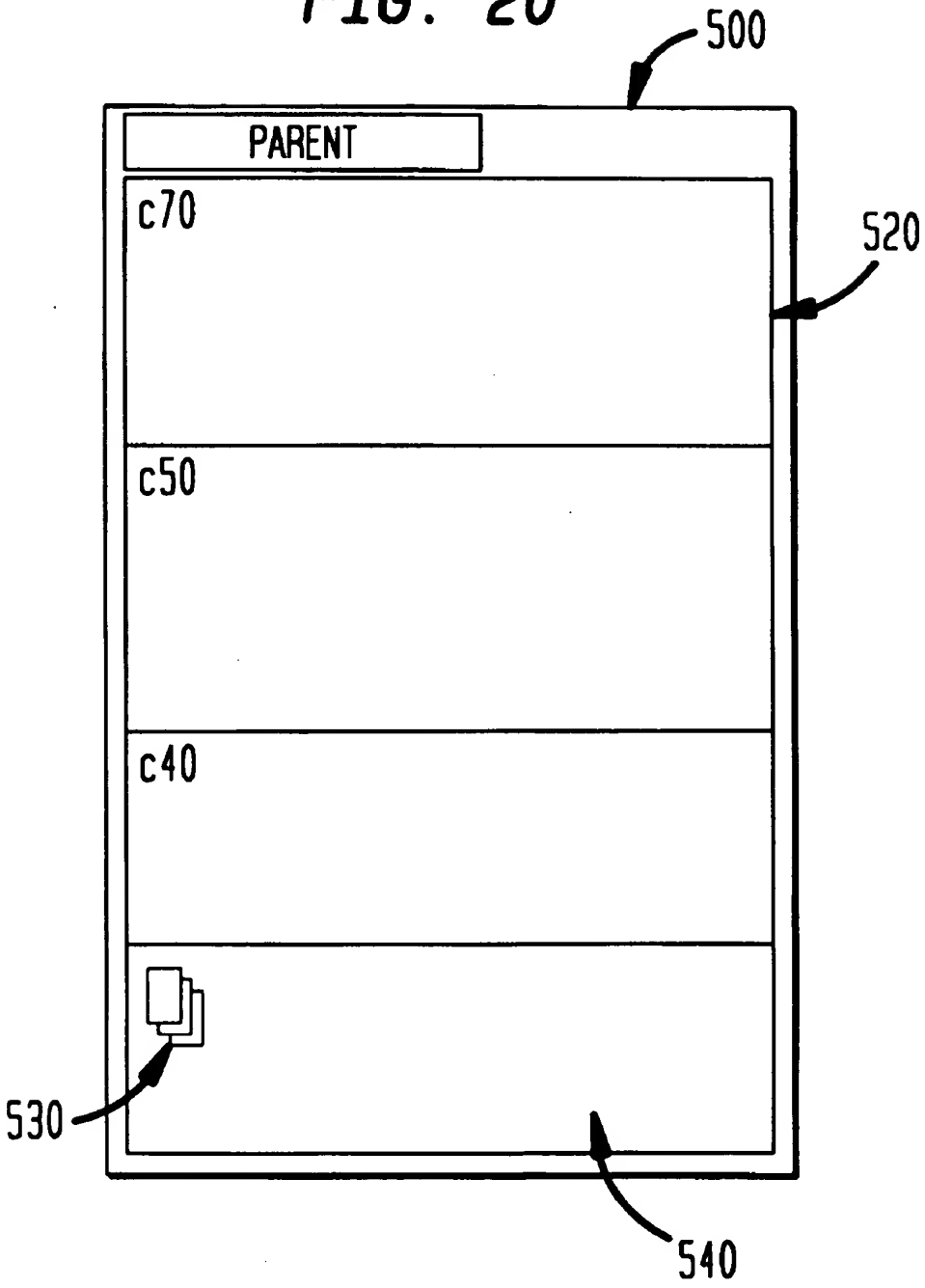
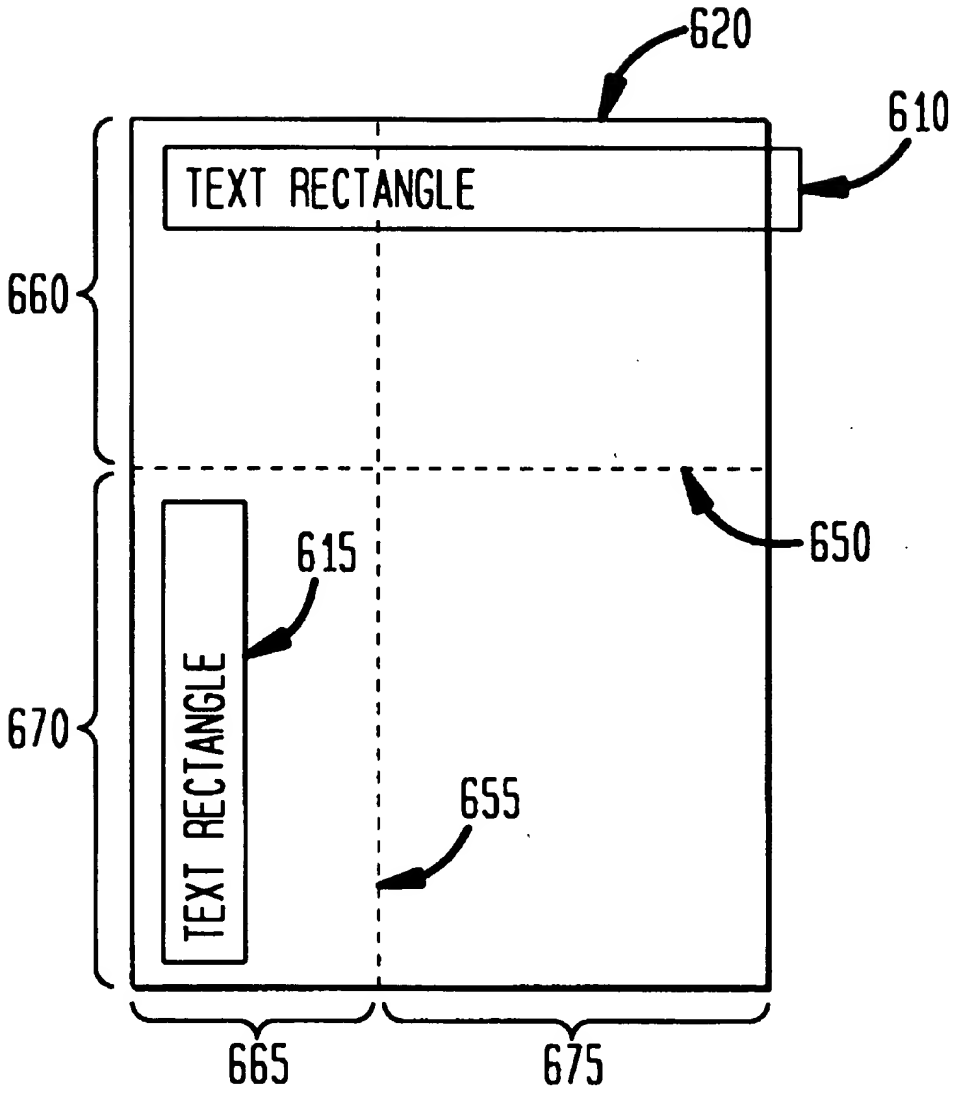


FIG. 21


620
610
TEXT RECTANGLE
660
650
615
TEXT RECTANGLE
670
655
665
675

METHOD AND APPARATUS FOR GRAPHICALLY REPRESENTING INFORMATION STORED IN ELECTRONIC MEDIA

CROSS REFERENCE TO RELATED APPLICATIONS

The present application claims benefit of U.S. Provisional Application No. 60/017,400 filed May 14, 1996, the disclosure of which is hereby incorporated by reference herein.

BACKGROUND OF THE INVENTION

A common problem encountered by computer users is running out of storage space on media such as hard disk drives on personal computers. The need for additional storage space is increasing as more and more software is being sold on high density media such as CDs, and more and more information is being downloaded from on-line services such as the Internet. All of this contributes to a computer's primary magnetic media storage devices filling up faster than ever. Even with the use of data compression utilities, storage space fills quickly.

Concomitant with the need to find more storage space is the need to manage the massive amount of data being stored on computers. Users need to clear space for new data and programs. Thus, in order to make room for additional data, there is a need for a tool or method which can help a user efficiently and easily identify the best data to be removed.

Many prior art file managers like Microsoft Windows File Manager or Windows Explorer simply illustrate the tree-structure of the directories and files on the computer. These file managers do not allow the user to quickly, easily and graphically determine what files and directories of files are unnecessarily taking up too much space.

The WinSurfer product developed by Ben Shneiderman, Brian Johnson, David Turo and/or Marko Teittinen and belonging to the University of Maryland teaches the use of nested rectangles to illustrate the tree-structure and relative sizes of files and directories. Each directory or file is represented by a rectangle whose size is proportional to the size of the directory or file. If a particular directory contains other directories or files, then those other directories or files are shown as rectangles within the rectangle associated with the particular directory. Because the program apparently simply divides rectangles, many of the rectangles are very long and thin which makes it difficult to gauge the size of the rectangles in relation to one another. The product's failure to control the dimensions of the rectangles also makes it difficult to annotate the rectangles with meaningful information such as text. Further, the shapes of the rectangles are not chosen to maximize the number of rectangles displayed.

There is, therefore, a need for a file management system which addresses the disadvantages of the prior art.

SUMMARY OF THE INVENTION

The present invention addresses this need.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a sample screen output in accordance with the present invention.

FIG. **2** is a schematic of an apparatus for implementing the present invention.

FIG. **3** is a representation of directories and files capable of being displayed in accordance with the present invention.

FIG. **4** is a representation of directories and files capable of being displayed in accordance with the present invention.

FIG. **5** is a representation of directories and files capable of being displayed in accordance with the present invention.

FIG. **6** is a schematic representation of a tree data structure to be displayed in accordance with the present invention.

FIG. **7** is a schematic representation of a tree data structure to be displayed in accordance with the present invention.

FIG. **8** is a schematic representation of a Specified Rectangle and its components in accordance with a preferred embodiment of the present invention.

FIG. **9** is a schematic representation of dividing a Specified Rectangle into a First Rectangle and Rest Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **10** is a schematic representation of determining whether a Text Rectangle and Expanded Rectangle fit within a Live Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **11** is a schematic representation of a Specified Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **12** is a schematic representation of a tree data structure to be displayed in accordance with the present invention.

FIG. **13** is a schematic representation of determining whether a Text Rectangle and Expanded Rectangle fit within a Live Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **14** is a schematic representation of grouping a tree data structure in accordance with the present invention.

FIG. **15** is a schematic representation of dividing a Specified Rectangle into Group Rectangles.

FIG. **16** is a schematic representation of a Specified Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **17** is a schematic representation of a Specified Rectangle and child rectangles within the Specified Rectangle in accordance with a preferred embodiment of the present invention.

FIG. **18** is a schematic representation of a tree data structure to be displayed in accordance with the present invention.

FIG. **19** is a schematic representation of a tree data structure in accordance with the present invention.

FIG. **20** is a schematic representation of a Specified Rectangle in accordance with another preferred embodiment of the present invention.

FIG. **21** is a schematic representation of a determining placement of Text Rectangle in accordance with yet another preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. **1** represents a sample screen output in accordance with the present invention. An apparatus for implementing the present invention preferably includes a general computer as shown in FIG. **2**, such as a personal computer. The computer may include the processing unit **57**, keyboard **51**, monitor **56** and screen **55**. Potential storage media include hard drive **52**, CD-ROM **53** or floppy disk drive **58**. Screen **55** will be used in accordance with the present invention to graphically illustrate the contents of the storage media.

As shown in FIG. **1**, screen image **10** includes a menu bar **36**, toolbar **37**, status bar **35**, traditional file tree **30**, and rectangle viewing area **20**. Rectangle viewing area **20** shows the contents of a selected storage medium, such as the directories and files on hard drive **52** of computer **50**.

Each directory and file is represented in the viewing area **20** as a rectangle, such as rectangle **40** which represents the root directory "c:" of the hard drive. The rectangles are nested to illustrate the directory/subdirectory/file manner in which information is stored on the hard drive. For example, as shown in traditional file tree **30**, the ART directory includes a number of other directories such as MISC, Letter mass and CRT. As shown in rectangle viewing area **20**, the directories MISC, Letter mass and CRT are represented by rectangles which are all nested within the rectangle labeled ART. Likewise, the files stored within the Letter mass directory, such as d13n.cdr, are represented as rectangles within the Letter mass rectangle.

The size of the rectangles are proportional to some characteristic of the corresponding directory or file, such as the amount of space occupied by the directory or file on the hard drive. For example, as shown in traditional file tree **30**, the size of the MISC directory and all of its subdirectories and files is about 10.4 megabytes. The size of the WELCOME directory, on the other hand, is roughly half that at about 5 megabytes. Thus, in viewing area **20**, the area occupied by the WELCOME rectangle is roughly half that of the MISC rectangle. Moreover, since the MISC directory occupies 10.4 of the 61.7 megabytes of the ART directory, the MISC rectangle occupies about 10.4/61.7 or about 16% of the space occupied by the ART directory. Because files do not contain other directories or files, the rectangles associated with the files do not contain other rectangles.

Each rectangle also displays additional information about its corresponding directory or file. For example, the name of the directory or file is displayed in each rectangle. The color of the rectangle may also convey additional information about the file such as its nesting depth.

As the cursor **39** is moved across screen display **10** by mouse **54**, information about the rectangle currently under the cursor will be displayed in status bar **35**. This information may include the file path, size, date, time, extension type and optionally any other details associated with the corresponding file or directory. The currently selected rectangle is highlighted such as by the use of diagonal bars **38**. If the user is interested in a particular directory or file which he wishes to delete or compress, then the user may issue a simple command to accomplish this task or click one of the corresponding buttons on toolbar **37**. The invention further allows a user to view or launch files by double clicking on the corresponding rectangle.

The method of generating rectangles in accordance with the present invention generally includes the steps of tiling the viewing area **20** with nested rectangles whose areas are proportional to the size of the files and subdirectories found in the first level (root) directory of the storage device. The dimensions of those rectangles are chosen in accordance with certain criteria. These rectangles are then tiled with the second level of directories and so on.

One preferred embodiment of the present invention includes the source code attached herewith and some of the major operations and steps performed by the source code are described below. However, for the sake of clarity and ease of understanding, the following description may deviate somewhat from the exact procedures and operations of the source code, particularly with respect to the order in which the steps are carried out.

1. Creation of tree data structure

In order to efficiently tile viewing area **20**, a tree data structure is created to represent all of the directories and files of the program. As popular personal computer platforms already store directories and files in a branched format, the data structure is modeled on that format. By way of example, FIG. **3** shows the directories and files on a hypothetical hard drive. For ease of reference, the name of each directory begins with the prefix "SUB" and the name of each file begins with the prefix "FILE". Each file is stored under at least one directory. Directories, in turn, are either stored directly under the root directory c: or are stored under one or more of the other directories. The numbers in parenthesis next to the files listed in FIG. **3** indicate the relative sizes of the files. Because directories are really pointers to other files and directories, it does not take much disk space to store the directories themselves as compared to the files they contain. Therefore, while the total amount of space occupied by the files under a directory may be quite large, the amount of storage space used to store the name of the directory is generally nominal.

Each directory and file is considered to represent one "node" of the tree data structure. In the case of directories, the node may contain many other nodes (in other words, files and other directories). If a first node contains one or more other node(s), then the first node is considered the "parent" of the one or more "child" node(s). Thus, node SUB B is the parent of child nodes SUB BA and SUB BB, and the node SUB BA is the parent of child nodes SUB BAA and FILE BA1.

When the directories and files are stored in the tree data structure, the sizes of the files are stored as well. As the tree data structure is created, the program recursively calculates the total size (either logical or physical) of all the files under any particular node and stores the value in the tree. These sizes and any other information the user might use (including the names of the directories or files) are also stored in the tree data structure. For example, as shown in FIG. **4**, each node includes the sum total of the size of all of the files under that directory.

Once the sizes of the nodes have been stored, the tree data structure is sorted so that the first node is the largest. In other words, and as shown in FIG. **5**, the nodes of the tree are sorted from largest to smallest which assists the program later on in determining how to tile.

To illustrate further steps of the method of the present invention, it is assumed that the invention will display the nodes under the Parent node shown in FIG. **6**. The Parent Node has seven children under it, namely **c10**, **c9**, **c8**, **c7**, **c6**, **c5** and **c4**. Some of these children are directories, namely **c10**, **c8**, **c7**, **c6** and **c4**. The remaining children are files, namely **c9** and **c5**. The children which are directories also have children. For example, **c10**'s children are **g4**, **g3**, **g2** and **g1** and **c8**'s children are collectively referred to as **g8** (the prefix "g" refers to the fact that these children are grandchildren of the parent node). The grandchildren may in turn be directories and have other children as well, such as **g8**'s children which are collectively referred to as **gg8** (the prefix "gg" refers to the fact that these children are great-grandchildren of the parent node). The sizes of the nodes **c4–c10** and **g1–g4** are assumed to be equal to their reference numbers. For example, the total size of node **c10** and all of its children is assumed to be 10 units, the total size of node **c9** is assumed to be 9 units, the total size of node **c8** and all of its children is assumed to be 8 units, and the total size of node **g4** is assumed to be 4 units.

The program travels through the tree to create the tiled images, and uses a pointer to keep track of the node it is

currently working on. This node is called the "Current Node" and when the program starts, the parent node, and thus the current node, will point to the root directory of the storage medium. The Current Node pointer is illustrated in FIG. **7**.

2. Creation of First Specified Rectangle

Each time the program points to a new current node, the program determines the best location and dimensions of the rectangle used to illustrate that node in viewing area **20** (FIG. **1**). This rectangle will show all the information associated with that particular node and is called the "Specified Rectangle." A sample Specified Rectangle **100** is shown in FIG. **8**. When the program starts at the root node, the location and size of the Specified Rectangle **100** will be set equal to the shape of the viewing area **20**.

3. Creation of Text Rectangle

Once the location and dimensions of the Current Node's Specified Rectangle are determined, a Text Rectangle **110** is created within the Specified Rectangle. The Text Rectangle **110** is used to display the name of the node associated with the Specified Rectangle **100** and its top-left corner **112** is placed adjacent to the top-left corner **102** of the Specified Rectangle. For optimal speed, the dimensions of the Text Rectangle are set equal to the estimated typical width and height of the names of the directories and files. For example, most if not all of the directories and file names in a Windows 3.11 platform are no greater than 12 characters long. Therefore, the dimensions of the Text Rectangle are preferably set equal to the height and width of a 12-character long word at a specific font size. Alternatively, the Text Rectangle could vary with the size of the Current Node's actual name.

If the Text Rectangle is too large to fit in the Specified Rectangle, then the name of the Current Node will either be partially displayed in the Specified Rectangle (i.e., "clipped") or not at all.

4. Creation of Live Rectangle

If the Current Node has children, then it is a directory and the rectangles of the Current Node's children will be displayed within the Specified Rectangle. Otherwise, if the Current Node has no children, then the Current Node is either a file or empty directory and, therefore, no children will have to be displayed in the Current Node's Specified Rectangle.

If the Current Node has children, a Live Rectangle **120** is used to display the rectangles associated with the children of the Current Node. In other words, a portion of the Specified Rectangle is assigned or allocated to show the rectangles associated with the children. Once the location and dimensions of the Specified Rectangle are known, the location and dimensions of the Live Rectangle are also known.

Specifically, the location and dimensions of the Live Rectangle are calculated so that there is a small margin **140** between the two side and bottom edges of the Live and Specified Rectangles. The margin **130** between the tops of the Live and Specified Rectangles is made large enough to accommodate the Text Rectangle. These margins help the user see the nesting of the directories and files under the Current Node's directory.

However, while margins are preferred to make it easier to view the rectangles, they are not necessary. Thus, when it is said that the height and width of a rectangle inside the Specified Rectangle is less than the height and width of the Specified Rectangle, this is intended to mean that the heights and widths would have been essentially equal but for the presence of margins which extend around and within the perimeter of the Specified Rectangle. Likewise, when it is said that the height and width of an inside rectangle is less than the height and width of the Specified Rectangle, the difference in heights and widths is more than just the space occupied by the margin. rather there is some additional distance between the inside rectangle and margins (which may be occupied by a different inside rectangle as explained below).

Once the location and dimensions of the Current Node's Live Rectangle are determined, the program proceeds to determine how to best tile the Live Rectangle with the rectangles of the Current Node's children.

If the current node has children, then the program also creates "First" and "Last" pointers to point to the first and last child of the parent as shown in FIG. **7**.

5. Preliminary Division Of Live Rectangle

To determine how the Live Rectangle should be divided into separate pieces for each of the children, the program will initially assume that the first child is going to be shown in the top portion of the Live Rectangle. Specifically, the program calculates the proportion of the size of the first child to the current parent node and divides the Live Rectangle accordingly. Using the example of FIG. **7**, this proportion is equal to the size of node **c10** divided by the size of the parent node, i.e. 10/(10+9+8+7+6+5+4)=10/49 or about 20%. As shown in FIG. **9**, the program essentially draws a horizontal line **150** which divides the Live Rectangle **120** into two rectangles, namely a First Rectangle **160** and a Rest Rectangle **170**, such that the area of First Rectangle **160** is 20% of the area of Live Rectangle **120**. The rectangles **160** and **170** are so-named because the First Rectangle **160** will show the first child **c10** and the Rest Rectangle **170** will show the rest of the children **c4–c9**.

6. Text Fit Testing

Once the First Rectangle is initially configured, the program looks to see how well the name of the first child will fit in the First Rectangle. If the text fits well, then the First Rectangle will become the first child's Specified Rectangle and the program will move on. Otherwise, if the text does not fit well, then the program will not use the First Rectangle to show the first child.

To test how well the text description fits, the program first determines whether the Text Rectangle **162** fits in First Rectangle **160**, as shown in FIG. **10**. (As used herein, one rectangle is said to "fit" in another rectangle when the height and width of the one rectangle are equal to or smaller than the height and width of the other rectangle.) The program next determines whether an Expanded Rectangle **164**, which is preferably 1.5 times the width of the Text Rectangle **162**, fits within the First Rectangle **160**.

Depending on whether or not the Expanded Rectangle fits within the First Rectangle, either the "booking" or "binary" method will be used. Specifically, if the Text Rectangle **162** fits in the First Rectangle **160** but the Expanded Rectangle **164** does not (FIG. **14**), then the "booking" method will be used to tile the first child into the Specified Rectangle. Otherwise, the "binary" method will be used for the children. Both methods are explained below.

7. Booking Method

If the Text Rectangle fits within the First Rectangle but the Expanded Rectangle does not, then the description of the first child will fit rather snugly within the First Rectangle. In other words, the First Rectangle **160** is not much wider nor thinner than the text to be displayed in the rectangle. This is called the "booking" method of tiling because, like most books, the title is generally the same size or a little shorter than the length of the spine. Accordingly, the look and dimension of the rectangles may be adjusted by changing the dimensions of the Text Rectangle and ratio of the Expanded Rectangle to Text Rectangle.

Using the booking method, the First Rectangle **160** will simply become the Specified Rectangle **180** of the first child c10 (FIG. 11). Further, since the First Rectangle is now occupied by the first child, the Rest Rectangle **170** of the Live Rectangle **120** (FIG. 9) is now ready to be tiled by the rest of the children. Specifically, the new Live Rectangle **186** (FIG. 11) is redefined to be equal to the previous Rest Rectangle **170** (FIG. 9). The program then moves on to the next child of the parent, i.e. c9. As shown in FIG. **12**, the program accomplishes this by moving the First pointer from the previous first child c10 to the next child c9. The program then performs the same process on c9 through c4 as it did on c10 through c4, starting at the "Preliminary Division of Live Rectangle" step.

The program will continue creating Specified Rectangles for all of the children of the parent node and moving the pointers accordingly. Finally, after the last child node has been processed, the program will know it is done with the children of the Current Node because the First pointer and Last pointer will point at the same node.

8. Binary method

As mentioned above, the booking method is not always used to tile. Rather, if the Expanded Rectangle does fit in the First Rectangle or if the Text Rectangle does not fit in the First Rectangle, then the "binary" method is used. Essentially, the binary method divides the Live Rectangle in two along its longest side and places the children in these two new squarish-rectangles.

For example, assume that the Specified Rectangle of the parent node does not have the shape shown in FIG. **12**, but is rather substantially wider and shorter as shown in FIG. **13**. In this instance, during the "Preliminary Division Of Live Rectangle" step, the Live Rectangle **220** will be divided at line **250** so that the First Rectangle **260** of node c10 still occupies 20% of the area of the Live Rectangle **220**. As is apparent, First Rectangle **260** of FIG. **17** is much wider and shorter than First Rectangle **160** of FIG. **10**. Thus, during the "Text Fit Testing" step, the program will determine that Text Rectangle **262** and Expanded Rectangle **264** both fit within First Rectangle **260** (in contrast to FIG. **10** where Expanded Rectangle **164** did not fit within First Rectangle **160**). Accordingly, First Rectangle **260** is not considered to be a good fit for Text Rectangle **262** because the First Rectangle is too wide and, therefore, First Rectangle **260** will not be used to display first child c10.

Therefore, the program splits the Live Rectangle **220** in two before it is tiled with the children. In the binary method, the children of the parent node are split into two groups, with all of the larger nodes being placed in one group and all of the smaller nodes being placed in the second group. The point of division is chosen so that the sum of the node sizes in the first group is as equal as possible to the sum of the node sizes in the second group. As shown in FIG. **14**, child nodes c10, c9 and c8 will be put in the first group **310** having a total size of 10+9+8=27 and child nodes c7, c6, c5 and c4 are placed in a second group **320** having a total size of 7+6+5+4=22.

As shown in FIG. **15**, once the two groups are created, the longer side of the Live Rectangle **220** is divided at line **255** to form two rectangles **315** and **325** which reflect the respective sizes of groups **310** and **320**. The area of the first group rectangle **315** is 27/(27+22) or about 55% of the total area of the Live Rectangle **220**. By dividing the Live Rectangle at the longer side, the group rectangles are more square than long and thin.

Each group rectangle **315** and **325** is then tiled with the nodes of that group. The program does this by temporarily changing the First and Last Pointers to point to the first and last pointers of the group to be tiled. As shown in FIG. **14**, if the first group **310** is processed first, then the new First' pointer will point to node c10 which is the first child in the first group **310** and the new Last' pointer will point to the last child in the first group which is node c8. Moreover, the Live Rectangle is now set to be equal to the first group's rectangle **315**. The program then repeats the process over again starting from the "Preliminary Division Of Live Rectangle" step as if the parent node only had three children, i.e., the children of the first group. The program continues until all of the nodes in the first group are tiled. In this fashion, all of the nodes of the first group will be tiled into the first group rectangle.

Once all of the nodes of the first group **310** are tiled into its rectangle **315**, the nodes of the second group **320** will then be tiled into their rectangle **325**. Specifically, a new First" pointer is set to point to node c7 which is the first child in the second group and a new Last" pointer will point to node c4 which is the last child in the second group (FIG. **14**). The Live Rectangle is then set to be equal to the second group's rectangle **325**. The program then repeats the above process starting from the "Preliminary Division Of Live Rectangle" step as if the parent node only had the children contained in the second group, and continues until all of the nodes in the second group are tiled. Once the second group **320** has been tiled into the second group rectangle **325**, the Live Rectangle **220** of the Specified Rectangle **200** of parent node will have been filled in with all of its children's rectangles.

The simplest case is where a group only has one child. In that instance, the Specified Rectangle of that child will simply be set equal to the entire group rectangle. By way of example, if node c10 was the only member of group **310**, then c10's Specified Rectangle would simply be set equal to group rectangle **315** and the second group would be processed next.

In an alternative preferred embodiment, the Live Rectangle could be split into more than two pieces if the circumstances warrant. For example, if the Live Rectangle were three times as long as it is high and the nodes could be split into three relatively equal groups, the Live Rectangle could be split into three group rectangles which would be more square than dividing the Live Rectangle into two pieces.

9. Combinations of Booking and Binary

FIG. **16** illustrates how the two groups **310** and **320** may be tiled into Specified Rectangle **200**. It will be noted that in the examples of FIGS. **17–20**, the binary method was initially used to divide Live Rectangle **220** of the parent node into two group rectangles **315**, **320**. In the scenario shown in FIG. **16**, the two group rectangles had dimensions which enabled the booking method to be used to fill up the group rectangles.

Thus, when the viewing area **20** is tiled with all of the various parent, child and grandchild nodes, it is likely that the binary and booking methods are both used in various combinations throughout the viewing area. Moreover, although the example of FIGS. **13–16** had an instance where the binary method was used once and then immediately followed by the booking method, that will not always be the case. It may well happen that the binary method does not make group rectangles which are automatically the right shape for using the booking method. If so, the binary method will be used to split the group rectangles and the process will continue.

Accordingly, the present invention tiles the child rectangles in accordance with two different styles depending on

how the Text Rectangle fits in the proposed child rectangle. If the Text Rectangle does not fit well, then the parent's rectangle is first split into two pieces such that any given child rectangles does not take up a whole side of the parent, i.e., both the height and width of the child are smaller than respective height and width of the parent. On the other hand, if the Text Rectangle does fit well, then the child rectangle will be placed in the rectangle such that the width of the child is about equal to the width of the parent and/or the height of the child is about equal to the height of the parent. For the purposes of this disclosure, the height and width of a child rectangle is considered equal to the height and width of the parent rectangle if the only difference between the two rectangles is the presence of margin such as margins **130** and **140**.

For example, it has been determined that when the average ratio of the longest side to the shortest side of a rectangle is 3, this ratio is particularly suited to display the nodes. Moreover, when using the parameters and methods identified above, rectangles having this ratio or less will generally be achieved.

10. Recursion

The foregoing illustrations have not taken into account that many of the child nodes **c4–c10** have their own children. For example, as shown in FIG. **6**, node **c10** has grandchildren nodes **g1–g4** which must be tiled within **c10**'s Live Rectangle. The well-known programming method of recursion is used to draw the grandchildren and great grandchildren. The foregoing method was used to tile the child nodes **c4–c10** of the parent node. Likewise, the same method may be used to tile the grandchild nodes **g1–g4** because these nodes are simply children which have node **c10** as their parent. Similarly, great-grandchildren nodes **gg8** are the children of the grandchildren nodes **g8**.

The recursive step may be considered to occur anytime after the location and size of a Specified Rectangle has been created for any node. For example, assume that the program has reached a point where it has just determined the location and size of the Specified Rectangle for node **c10** of FIG. **6**. The binary and booking methods as described above state that the program will usually proceed to the next child **c9** once the Specified Rectangle of **c10** has been configured. However, before the program actually proceeds to child **c9**, the program will first determine whether the current child **c10** has children of its own. If so, the program will repeat the entire process over again starting at the "Creation of Text Rectangle" step as if **c10** were the parent node and **g1–g4** were the children. The only substantive difference will be that instead of setting the new parent node's Specified Rectangle to be the shape and size of the entire viewing area **20**, the location and dimensions of **c10**'s Specified Rectangle will be used. The program will not move on to child **c9** until Specified Rectangles have been created for all of **c10**'s descendants.

FIG. **17** illustrates the similarity between the Specified Rectangles **490** of node **c10** and the Specified Rectangle **400** of the parent node. Both have their own Text Rectangles **410** for displaying the names **482** of the nodes as well as Live Rectangles **420** for displaying the rectangles of the children of the nodes.

11. Hiding small files

Preferably, the invention includes a number of options which further increases the user's ability to easily view the contents of the storage media. For example, some rectangles may be only a few pixels large and too small to be perceived on the screen. The program addresses this problem by suppressing the display of nodes which are too small.

Generally, the user will set a minimum rectangle side which represents the smallest rectangle side which they wish to see on the screen. For example, the program may have a default minimum rectangle side so that any nodes having a rectangle with a side smaller than 5 pixels is suppressed (the "Minimum Square"). The program accordingly finds the smallest node which fits that Minimum Square size by finding the child node whose size as a proportion to the Current Node size is larger than the area of the Minimum Square as a proportion to the Live Rectangle size. All the child nodes which are smaller than the Smallest Child are marked as "Hidden." The program will not compute the Specified Rectangle of any node which is marked as hidden. However, in order to ensure that rectangle space is properly allocated to the hidden files, a new child node called "Small Files" is created. The size of the Small Files node is set equal to the sum of the children marked Hidden. Moreover, when the program encounters a Small Files node, the program will show a special icon in place of the name of the node to inform the user that that rectangle represents more than one directory or file.

Using the example of FIG. **18**, assume that the Live Rectangle of the parent node has an area of 100 pixels to display the rectangles of children **c70, c50, c40, c30** and **c10**. Further assume that the user does not want to see any Specified Rectangles which are less than 4x4 pixels, i.e. have an area less than 16 pixels. If so, then node's **c30** and **c10** will be marked as Hidden because the proportion of their size as compared to the parent (30/200=15% and 10/200=5%) is less than proportion of the size of the minimum rectangle to the Live Rectangle (16 pixels/100 pixels=16%). As shown in FIG. **19**, these nodes will be marked as "Hidden" and a new node called Small Files and having a size of 40 (which is equal to the sum of the sizes of **c30** and **c10**) will be inserted in sorted order as a child to the parent node. As shown in FIG. **20**, the non-hidden nodes **c70, c50** and **c40** will be tiled into the parent's Live Rectangle **520** as described above. A Specified Rectangle **540** will also be created for the Small Files Node, but icon **530** will be displayed instead of the node's name. No rectangles are created for hidden nodes **c30** and **c10**.

13. Vertical windows

As shown in FIG. **9** and as described in the "Preliminary Division Of Live Rectangle" step, the Live Rectangle **120** is initially split horizontally by line **150**. However, before the Text Fit Testing step occurs, the program will preferably try to determine whether a vertical split is better to show the name of the first child.

As described above in the "Preliminary Division Of Live Rectangle" step, the program first uses horizontal line **650** to divide the Live Rectangle **620** into a First Rectangle **660** and a Rest Rectangle **670**. If the Text Rectangle **610** can not fit in the Live Rectangle **620** when it is split horizontally, the program next determines whether the Text Rectangle could fit in the Live Rectangle vertically. As shown in FIG. **21**, the program does this by creating a vertical line **655** to split the Live Rectangle into a vertical First Rectangle **665** and a vertical Rest Rectangle **675**. The vertical rectangles have the same areas as their respective horizontal rectangles. The program next determines whether the Text Rectangle **615** could fit in the First Rectangle **665** vertically. If so, vertical First Rectangle **665** will become the First Rectangle, vertical Rest Rectangle **675** will become the Rest Rectangle, and the program will proceed to the Text Fit Testing step. The only difference will be that when the name of the child is displayed in the Text Rectangle **615**, the name will be shown rotated 90 degrees.

14. Colors

Preferably, the colors of the rectangles correspond with some attribute of the file or directory. For example, the file's or directory's color may change depending on the nesting depth. Alternatively, the color may change depending on the file's or directory's age or any other attribute of the corresponding file or subdirectory.

As mentioned previously, the foregoing steps broadly describe some of the major functions of the method of the present invention. These steps generally correspond with the routines of the attached source code as follows:

Step(s)	Routine
Creation of tree data structure	LoadSubdirectoryFromDisk; SortChildrenInSizeOrder
Preliminary Division Of Live Rectangle	ShowMapFilesScreen:PreliminaryDivision
Text Fit Testing	ShowMapFilesScreen:TextFitTesting
Booking Method	ShowMapFilesScreen:BookingMethod
Binary method	ShowMapFilesScreen:BinaryMethod
Recursion	ShowMapFilesScreen:ShowMapDirectory Screen; LoadSubdirectgoryFromDisk; SortChildrenInSizeOrder
Hiding small files	AddSmallFilesNode

The present invention provides a wide number of advantages. For example, the program displays directories and files of a storage medium in a graphical interface using nested rectangles. The size of each rectangle represents the proportional size of the directories or files and the nesting of each rectangle represents the branched-tree structure of the manner of storing the directory and file.

The present invention further tiles these rectangles in a manner which optimizes the readability and usefulness of the display. The invention balances the importance of a variety of factors, including: maximizing those number of rectangles which have dimensions capable of displaying the names of directories and files; promoting the creation of squarish rectangles so that the user can easily compare the relative sizes of two rectangles; minimizing the number of very long and thin rectangles; preventing directories or files from being shown if they are so small that their rectangles will clog the display with imperceptible information; and choosing colors which provide useful information. The program strikes a balance between all of these considerations.

The invention allows the user to easily see and identify which directories are taking the most space. Large or not-recently-used chunks of information can be identified, deleted, compressed or archived in order to free valuable storage space. The ease of being able to see where file space is allocated assists the user in avoiding the time, expense, and risk (such as to data) of buying and installing larger storage devices.

The invention further allows the user to see a map of the entire hard drive of a personal computer in the limited amount of space of a computer screen. Because of the manner in which the data is displayed, the invention is also a useful device for teaching computer novices about tree structured information.

The present invention has the further advantage of being able to be used with any number of different computers and storage mediums. For example, while the preferred platforms are Windows 3.x and Windows 95 on 386 or higher computers, the program may be adapted for Apple Macintosh and other systems. The invention can also be easily translated as an add-in inside a standard file manager such as the Windows 97 Explorer.

The program also does not exacerbate one of the problems it addresses: preventing the unnecessary use of extensive storage space. The attached executable version of the source code only occupies about 600 Kb of disk space.

The present invention a l so presents the advantage of utilizing a wide number of options. For example, in addition to showing the name of the file or directory in each rectangle, invention may allow the rectangle to alternatively show other information such as the age, access attributes or other useful information.

The program may further be configured to only display certain desired information. For example, a filter may be applied so that the invention only displays all directories and files satisfying a specific criteria, such as files having a specific extension. Even if only a few of the files are shown, those that are shown can still be displayed in a nested relationship to illustrate the files' respective directories. Other conditions may also be applied, and these conditions may be applied not just at the file level, but at the directory and subdirectory level as well. These conditions may include whether a name starts with a certain letter, when the file or directory was last accessed and how big the file or directory is. Another filter would only show files for which there are duplicates (another feature would show all the files but draw a line between the duplicates). The subroutine RecalcFilter of the attached source code represents one preferred embodiment of this method.

The program may also be configured so that the user may choose to only show the directories and files under a particular directory as compared to the entire storage device. In other words, for any selected directory, the program may zoom in so that the selected directory will fill the entire viewing area **20** for closer examination. A zoom-out command is also provided so that the parent of the selected file or directory fills the entire viewing area **20**. The program would perform the zoom-in command by using the steps described above, but would substitute the target directory for the root directory. When zoomed-in, the user may also have the option of showing an outline of the currently selected rectangle as if the directories of the root directory were shown, so that even though the user has enlarged all of the rectangles of the currently selected directory, the user can still see the relative size of the currently-selected rectangle compared to the entire disk drive.

Another feature includes advising the user which files are ripe for deletion (i.e., not recently accessed and large data files) and which the user should be cautious in deleting (i.e., executable versions of programs). Yet another feature would allow the user to select the depth of nesting shown. For example, the user could choose to show only one level of nesting so that only the directories and files immediately under the root directory are shown. In yet a further option, the rectangles could show a representation of the data contained in the associated file, such as showing a portion of the text when the file represents a document or a picture when the file represents a picture.

The invention also serves as a standard file manager for copying, moving, and otherwise managing files. For example, as represented in the attached source code, the user may select which subdirectory is to be mapped, select individual files, show information about files that are too small to see on the map drag files to and from the map and perform other file management operations.

The program may also be modified to continuously monitor the amount of free space on the disk drive so the user can be warned when disk space is lower than a value or percentage specified by the user. This helps users avoid error

conditions that result from running out of free space. In this instance, the program could run in the background on start-up so that the program is constantly monitoring disk space even if the user is not looking at the program's screen.

Moreover, it is not required that the nodes of the tree data structure be displayed rectangles. For example, the program could be modified so that it displays ellipses which, of course, have heights and widths. Other shapes such as circles are also possible.

Another advantage of the present invention is that it has wide application beyond those showing files and directories on a storage medium. The invention can be used to graphically represent any tree data structure stored in some medium where the nodes contain numerical information. For example, a city could be represented by neighborhoods, streets and houses where the numerical quantity is acreage. Another application includes displaying Internet link connections, where each node represents a web page and the children represents links to other web pages. The numerical quantity associated with the node pages could vary with, for example, with the amount of data contained by the web page. Alternatively, the numerical quantity associated with the "leaves" of the tree data structure (i.e., nodes having no children) could be set to a constant such as one and the nodes of the tree with children would be set equal to the sum of their descendants. If so, the size of the associated rectangles would be proportional to the number of the descendants. Database tables which were converted to trees could also be displayed.

Although the invention herein has been described with reference to particular embodiments, it is to be understood that the embodiments are merely illustrative of the principles and application of the present invention. It is therefore to be understood that numerous modifications may be made to the embodiments and that other arrangements may be devised without departing from the spirit and scope of the present invention as defined by the claims.

We claim:

1. A computer implemented method of displaying information stored in a tree data structure stored in a computer storage medium, comprising:

(a) providing a tree data structure comprising nodes, said data structure having a relationship whereby any one node may contain one or more other nodes, each said node having a numerical quantity associated with a node, whereby said numerical quantity associated with said node is proportional to the amount of space occupied on said computer storage medium,

(b) providing a parent node in said data structure,

(c) providing a child node in said data structure, said child node being contained in said parent node,

(d) displaying said parent node on an output medium as a parent shape having a height, width and area, the area of said parent shape being proportional to the numerical quantity associated with said parent node,

(e) displaying said child node on an output medium as a child shape having a height, width and area, the area of said child shape being proportional to the numerical quantity associated with said child node, and

(f) said height of said child shape being less than said height of said parent shape and said width of said child shape being less than said width of said parent shape.

2. The method of claim 1 wherein said nodes represent directories and files stored on a computer storage medium and said numerical quantity associated with a node is proportional to the amount of space occupied by the associated directory or file on said computer storage medium.

3. The method of claim 1 wherein said nodes represent links to sites on a network.

4. The method of claim 3 wherein said network is the World Wide Web, said nodes represent web pages containing links to other web pages, the numerical quantities of nodes without children are equal to one another, and the numerical quantity of nodes with children is equal to the sum of the numerical quantities of the nodes contained by the node with children.

5. A computer implemented method of displaying information stored in a tree data structure stored in a computer storage medium, comprising:

(a) providing a tree data structure comprising nodes, said data structure having a relationship whereby any one node may contain one or more other nodes, each said node having a numerical quantity associated with a node, whereby said numerical quantity associated with said node is proportional to the amount of space occupied on said computer storage medium,

(b) providing a parent node in said data structure,

(c) providing a child node in said data structure, said child node being contained in said parent node,

(d) displaying said parent node on an output medium as a parent shape having a height, width and area, the area of said parent shape being proportional to the numerical quantity associated with said parent node,

(e) assigning a portion of said parent shape, at least a portion of said assigned portion for displaying a shape associated with said child node,

(f) determining whether the width of said assigned portion is greater than a first predetermined amount and less than a second predetermined amount,

(g) if said width of said assigned portion is less than a first predetermined amount or greater than a second predetermined amount, then displaying said child node on an output medium using a binary display method, and

(h) said binary display method comprising displaying said child node as a child shape within said assigned portion, said child shape having a height, width and area, the area of said child shape being proportional to the numerical quantity associated with said child node and said height of said child shape being less than said height of said parent shape and said width of said child shape being less than said width of said parent shape.

6. The method of claim 5 wherein said medium is a computer storage medium, said nodes represent directories or files, and said numerical quantity associated with a node is proportional to the amount of space occupied by the associated directory or file on said computer storage medium.

7. The method of claim 5 wherein the width of said assigned portion is about equal to the width of said parent shape and the height of said assigned portion is less than the height of said parent shape.

8. The method of claim 7 wherein if said width of said assigned portion is greater than said first predetermined amount and less than said second predetermined amount, then said method further comprises displaying said child node on an output medium using a booking display method, said booking display method comprising displaying said child node on an output medium as a child shape having a height, width and area equal to the height, width and area of said assigned portion.

9. The method of claim 8 wherein said step of determining whether the width of said assigned portion is greater than said first predetermined amount and less than said second predetermined amount comprises

(a) providing a text shape having a width,

(b) computing an expanded shape having a width larger than the width of said text shape,

(c) determining whether the width of said text shape is less than the width of said assigned portion,

(d) determining whether the width of said expanded shape is greater than the width of said assigned portion.

10. The method of claim **9** wherein the width of said expanded shape is about 1.5 times the width of said text shape.

11. The method of claim **5** wherein, if the width of said assigned portion is less than a first predetermined amount or greater than a second predetermined amount, then said binary method further comprises dividing said assigned portion into a first group shape and a second group shape each having a width, height and area, such that said width of said group shapes is less than said width of said assigned portion and said height of said groups shapes is about equal to said height of said assigned portion and displaying said child shape in one of said group shapes.

12. The method of claim **11** wherein said data structure further comprises a secondary node contained in said parent node but not contained in said child node, and further comprising displaying a shape associated with said secondary node in said second group shape.

13. The method of claim **12** further comprising

providing additional nodes contained in said parent node but not contained in said child node or secondary node,

allocating each of said child, secondary and additional nodes into a first group or second group such that the smallest numerical quantity associated with a node in said first group is greater than the largest numerical quantity associated with a node in said second group, and

displaying said nodes of said first group as shapes in said first group rectangle and displaying said nodes of said second group as shapes in said second group rectangle.

14. The method of claims **5**, **6**, **8** and **13** wherein said shape is rectangular.

15. The method of claim **5** wherein said child shape is only displayed if said height and width of said child is greater than a minimum amount.

16. The method of claim **5** further comprising displaying text describing said parent node in said parent shape and displaying text describing said child node in said child shape.

17. The method of claim **16** wherein said text describes the name of a directory or file associated with said node.

18. The method of claim **16** wherein said text describing said parent node is displayed vertically within said parent shape.

19. The method of claim **16** wherein said text describing said parent node is displayed horizontally within said parent shape.

20. The method of claim **5** wherein said output medium is a computer screen.

21. The method of claim **20** wherein said height represents a vertical distance on said output medium.

22. The method of claim **20** wherein said height represents a horizontal distance on said output medium.

23. The method of claim S wherein substantially all of said nodes of said structure are displayed as shapes by recursively repeating said steps of said method as if each node containing other nodes is said parent node.

24. The method of claim **5** wherein said steps of displaying further comprises displaying said shapes as colors dependent upon an attribute of the node.

25. The method of claim **24** wherein said color of a shape associated with a particular node is dependent upon how many nodes contain said particular node.

26. A computer implemented method of displaying parent directories or files and child directories or files contained on a storage medium, whereby said child directories are contained within said parent directories, said method comprising:

(a) providing a computer screen having a resolution for displaying information stored in a tree data structure;

(b) displaying said parent directories in said tree data structure as rectangles, the area of each rectangle being proportional to the amount of space on the storage medium occupied by the associated parent directory;

(c) displaying said child directories in said tree data structure as rectangles, the area of each rectangle being proportional to the amount of space on the storage medium occupied by the associated child directory, and displaying the rectangles of said child directories within said rectangles of the parent directories which contain the child directories, and

(d) calculating the height and width of said rectangles so as to maximize the number of rectangles which can be displayed on said computer screen.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,987,469
DATED : November 16, 1999
INVENTOR(S) : James D. Lewis et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page,

Abstract,
Line 3, after "such" insert -- as --.

Column 6,
Line 3, "rather" should read -- Rather,--.
Line 52, "14" should read -- 10--.

Column 7,
Line 30, "12" should read -- 10 --.
Line 36, "17" should read -- 13 --.

Column 9,
Line 4, "rectangles" should read --rectangle --.

Column 10,
Line 30, after "than" insert -- the --.

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,987,469
DATED : November 16, 1999
INVENTOR(S) : James D. Lewis et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 12,
Line 5, "a l so" should read -- also --.
Line 8, after "rectangle," insert -- the --.
Line 62, "map" should read -- map --.

Column 13,
Line 19, "represents" should read -- represent --.
Line 21, cancel the word "with".

Column 16,
Line 15, "S" should read -- 5 --.

Signed and Sealed this

Thirty-first Day of July, 2001

Attest:

Nicholas P. Godici

Attesting Officer

NICHOLAS P. GODICI
Acting Director of the United States Patent and Trademark Office